AN ENHANCED ANALOGY BASED MODEL FOR THE EFFORT ESTIMATION OF SOFTWARE PROJECTS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

V SRINIVAS RAO
Y011414

DEPARTMENT OF INDUSTRIAL AND MANAGEMENT ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
Kanpur, India
February, 2002

25-2-02

Certificate

It is to be certified that the work contained in this report titled **"An Enhanced Analogy Based Model for the Effort Estimation of Software Projects"** is the original work carried out by V Srinivas Rao (Roll No. Y011414) under my supervision and has not been submitted elsewhere for a degree.

Veena Bansal

(Veena Bansal)
Supervisor
Department of Industrial and Management Engineering
I.I.T. Kanpur

This work is dedicated to my mother whose care and support I always feel through every thick and thin................

V Srinivas Rao

Acknowledgements

I take this as an opportunity to convey my deep sense of gratitude towards my supervisor Dr. Veena Bansal for her guidance, concern and meticulous attention throughout my tenure at I.I.T Kanpur.

I am extremely thankful to Mr. Rakesh Maheshwari, SEI Authorised CMM Trainer and Director, Ministry of Information Technology , Government of India , Ms Sunaina Banga ,HR Manager, NetBase Computing Noida, Mr. Avinash Tripathi , Project Manager, Infosys Ltd., Bangalore , Mr. Nitin Malik, Project Manager, TCS Delhi, and Mr. K.G. Mishra, CEO, QualityMeter.com for acquainting me with the practical scenario regarding product development in their organizations

I sincerely appreciate the co-operation of my batch mates in I.I.T. Kanpur, especially Mr. V Chandrashekhar and Mr.Vivek Pandey for providing me with the necessary information regarding coding in JAVA .

I heartily appreciate the company of all friends who made my stay in I.I.T. memorable.

I don't have words to express my gratitude towards my Mother whose care and support I always feel through every thick and thin.

V Srinivas Rao
I.I.T. Kanpur
February 25th , 2002

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Abstract

Estimation by Analogy is a technique that has been proposed since a long time as a valid alternative to algorithmic effort estimation and expert judgement. In this work we have enhanced the original model on estimation by analogy to search for the nearest two analogous projects. The features of these analogous projects are provided to the user for taking a decision on the new project. Our enhancement is in the terms of the number of project attributes that characterize a project. While scaling the factors, we have incorporated practical working environment prevailing in some of the CMM optimizing level organizations that we surveyed during the course of this work. We have also coded and tested four versions of the model to arrive at a conclusion that the Hot- Deck Imputation method is recommendable for dealing with the sparse data fields of the completed projects.

Through this thesis, we argue that no one method for effort estimation is a viable option. One should work on more then one estimating technique before arriving at some conclusion regarding the new project.

Chapter 1
Introduction

Last few decades have fully exploited the information technology to meet institutional, social and educational needs, and this trend is still going on. Software has become a critical issue in this modern civilization. Now everyone seems to need more and better software with tight schedules and cheaper rates. This is leading to new technology innovations and developments

With the advancement in the technology, the cost of hardware is consistently decreasing; on the other hand the cost of the software is increasing [3].

The main reason behind the high cost in software is that software technology is still labor intensive Software projects are very large, involving many people and most important part is that it is highly uncertain about schedule. To overcome these difficulties there is an urgent need for the proper software development methodologies and management techniques.

1.1 Overview

In general we can divide the software management techniques into two phases over the lifetime of a project (Jalote), the very first is the project planning and the second one is the project monitoring and control Broadly we can say that planning entails all activities that must be performed before starting the development of the project work. Once the project work is started project control begins.

1.1.1 Software Planning: A Management Perspective

Planning is perhaps the most important management activity and at the same time it is the weakest too. Many of the failures in software projects, caused by week management can be attributed to the lack of proper planning. But the software project management is different from the other engineering fields due to several factors [8]. These factors are specified in the following points:

1. Software is generally more complex than the other engineering products.
2. Since software engineering is relatively new, there are not yet many managers or the senior professionals with sufficient experience to appreciate the benefits of an effective process as compared to the conventional projects.
3. The insignificant cost of reproducing software forces software solutions to many late discovered system problems. While this is often the only practical approach, it further complicates the software job.
4. In the other engineering fields, release to manufacturing provides a natural discipline that is not present with software. When some independent party must sign off on production costs and schedules, the design definition is either complete or all work stops until it is.
5. The software is not grounded in the nature science. As an artifact of the human ingenuity, software does not rest on a stable foundation of physical principals
6. Software is often the system element that presents the function to the end user. It is thus the part that is the most visible, most subjects to complaint, and most exposed to requirement changes.
7. Software also is often crucial ingress client that couples all the other system elements together.
8. Non-software practitioners often view software as a black art. This causes many managers to back away and to use their management instincts to solve software problems.

Software project management is the collection of techniques used to develop and deliver a software. This developing discipline traditionally includes technical issues such as the choice of software development methodology, how to estimate project size and schedule, what resources to reuse, and which programming development environment to use. The discipline also includes management issues such as when to train personnel, what are the risks to the project success, and how to keep the project on schedule. These choices are then embodied in a software project management plan.

Software project management addresses both the process of software development and the desired functional characteristics of the final software product. A complete software project management plan is the design, implementation, control and test strategy for a software development process.

1.1.2 Few Models

Software development is complicated as it involves many people from different areas and with different skills, experiences and social attitudes. Many operational decisions are taken during this extended activity. There are many approaches to manage this complex activity However. Each approach addresses high level decisions process as well as detailed activities and their management. For instance, CMM/ISO 9000 series deal with high level decision process where as COCOMO/ PRINCE and FPA deal with detailed activities.

All of these methods have as a major function the attempt to anticipate and avoid all possibilities, which may negatively impact a software project. The negative possibilities are those which would delay the delivery of the software which performs the desired functions in a timely and cost effective manner. An additional function is to avoid late changes to the system because the later the change the more expensive it becomes. However, none of these methods consider the ethical issues that need to be identified and addressed during the planning stages and re-considered throughout the development process.

Effective software project management is a vital ingredient in achieving a successful outcome. The objectives for the project need to be agreed at the outset. In deciding the objectives their implications need to be considered, in terms of the actual outputs and the impact these outputs will have. There is also a need to consider the impact of the development process itself. The project team should be well briefed on these issues and have the opportunity to debate them fully to establish its own conclusions. The team should consider all the implications of the plan, including ethical ones. It may need to call on additional resources from inside and outside the organization. To confine the discussions within close boundaries (in an attempt to save money and time) is misguided. Broader issues will inevitably arise during the course of the project. If the team members are unprepared, they will lack direction and perform poorly. The sponsor of the project therefore needs the vision and the authority to ensure that the project team is supported and coached to consider both technical and ethical issues [Shneiderman, 1990, Huff and Jawer, 1994].

1.2 Importance of Effort Estimation in Software Engineering

Effort estimation is the most important and first step in the planning activity which gives the effort calculation for the development of the software to satisfy the given requirements and the time to complete the project. To estimate cost it's a primary need to estimate effort that would be needed to develop the project. The estimate of the cost enables the client or the developer to perform a cost benefit analysis.

For the team developing software a detailed and the accurate effort estimate for all different activities is invaluable for the project control, once development begins. Deviations of the actual cast efforts from effort estimate are used as a metric for accessing project states. In addition an effort estimate also gives the staffing levels for a project during different phases. Effort in a project is due to staff requirement for software, hardware and staff activities needed for the product development. Hardware resources are such things as the computer time, technical time and the memory required for the project, whereas software resource include the tool and compiler that are needed during the development. The major part of the cost is due to human factors, most of the cost model focus on this aspect.

1.2.1 Project Cost Estimation as a Management Requirement

Project management consists of three primary areas of responsibility, which are:

1. **Planning**:
   - Define Tasks
   - Schedule Tasks
   - Cost Tasks
   - Identify and Request Needed Resources
   - Document the Plan
2. **Monitoring**:
   - Cost Progress (expenditures vs. plan)
   - Schedule Progress (task completions vs. plan)
   - Quality
   - Technical Performance
3. **Reporting**:
   - Status (progress, accomplishments, problems, etc.)
   - Funds (allocated staff-hours)
   - Cost/Schedule Performance (indexed against plan)

Of these **planning** is arguably both the most important and most difficult, and especially for software development projects.

By definition, the **scope** of a project establishes a boundary around the work to be performed in accordance with the terms of the contract (or other project authorization). The manager **must** account for all *in-scope* work in the project's cost and schedule plans. This means that the manager must be able to accurately project schedule and cost requirements for all in-scope work.

Normally, the overwhelming cost and schedule driver for a software development project consists of the activities associated with designing, coding, and testing the software itself. There is also a direct (but non-linear) correlation between the amount of software to be written and the effort and schedule needed to produce it. This relationship is also affected by such factors as the complexity of the software to be developed, the reliability required of it in operation, the capabilities of the participating analysts and programmers, etc.

1.2.2 Uncertainties in Cost Estimates

In general when we deliver the product we know all about it cost. But when the project is initiated, during flexibility study, at the moment we know a little about the project and the data. Thus a large amount of uncertainty is associated about actual specification of the system. Specification with uncertainty represent a range of final product not one precisely defined product

As we reach towards final stages the more knowledge we have with us about the project and uncertainties are reduced and more accurate cost estimates can be made [9].

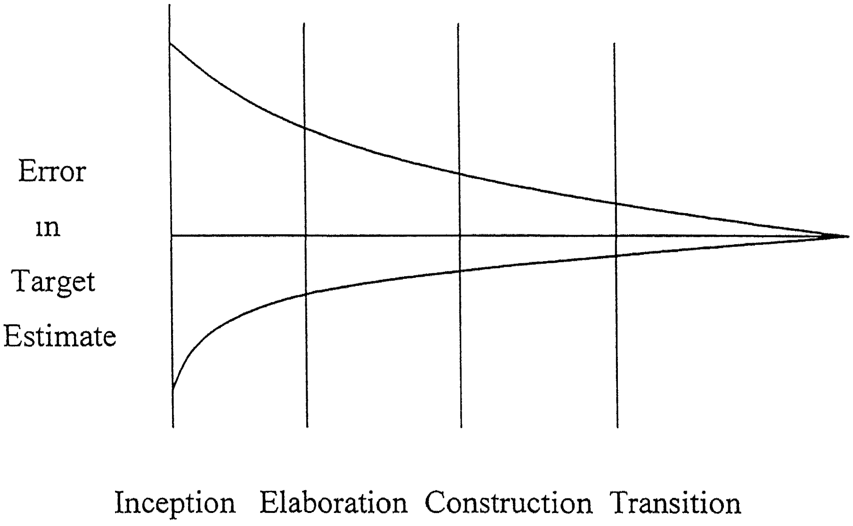


Fig. 1 1 Error in Target Estimate during Software Life Cycle

For actual cost calculations cost models have to be developed which can be used for estimation. The accuracy cost estimation will depends upon effectiveness and accuracy of the cost model employed.

1.3 Problem Statement

As we have already discussed, the challenging and the significant field of research in software engineering field is the investigation to reduce the development cost and keeps software projects with ın the schedule. This requires precise effort estimation models. There is certainly a need for developing a model that would enhance the precision of effort estimates at an early stage in software development.

The organizatıons with CMM repeatable level [Appendix I] and above have established plans for developing effort estimates. Possessing a level of CMM describes what we would normally expect in a software process in terms of the key processes practiced, regardless of how the process is implemented. This is why even the repeated and above level organizations have the scope to switch over from 'good' to 'better' planning. Therefore software organizations are in search for excellence in there key process areas, effort estimation being one of them, for which it is necessary to prepare a model that provides good effort estimates.

1.4 Our Approach

This thesis was inspired by the notion that accurate effort prediction must be an antecedent of producing lines of code. The general practices of effort estimation in software organizations are algorithmic models like COCOMO and most often by expert judgments.

Although software engineers and managers often know their problems in great detail, while estimating they face the problem of integration due to insufficient data available from the past history regarding actual effort for similar products developed in the organization. There are numerous factors that are responsible for the effort associated Our model works on the principle of estimation by analogy to arrive at the estimates for the new project. We have enhanced the work done by Shepperd [17]. Our enhancement is in terms of number of project attributes. We have also tested various methods to deal with the sparse data.

1.5 Organization of the Thesis:

Chapter two entitled **Literature Review** provides a reference for the cost evolution during the last years and also gives an introduction to Software Engineering Economics. It covers the different techniques for the effort estimation and at the same time it gives the detailed discussion of the effort estimation by analogy.
Chapter three entitled **The Proposed Model** covers in detail about our approach of effort estimation by analogy. It also discusses the need of the data set management
Chapter four entitled **Software Development** includes the Software Requirements Specifications document needed for coding the proposed model. It also includes a sample walk through which illustrates the various features of the software developed.

Chapter five entitled **Testing and Results** covers testing of various versions of the model. The results obtained are analyzed.

Chapter six discusses **Conclusion And Future Work.**

Appendix (I): specifies **Capability Maturity Model**

Appendix (II): specifies **Software Project Management Steps**

Chapter2
Literature Review

2.1 Background

The process of cost estimation started with the development of commercial software development around 1960s and 1970s, and that can be best described as the craftsmanship, with each project using a custom process and software tools. In the year 1980 and 1990, the software industry matured and transitioned to more engineering discipline, however, most software projects in this era were still preliminary research ıntensive, dominated by human creativity and diseconomies of the scale. The next generation of the software processes is driving towards a more production-intensive approach dominated by automation and economies of scale.

Many types of the research have investigated cost estimation. The purpose of this section is to review key studies from the research, which reinforce the importance of the topic and the frequent inaccuracy in estimation.

2.2 Software Engineering Economics

Most of the software cost models can be abstracted into a function of five basic parameters: size, process, personnel, environment, and required quality.

1. The size of the end product is typically measured in the number of source instructions or the number of function points required developing the required functionality.
2. The process used to produce the end product, in particular the ability of the process to avoid non-value-adding things.

3. The capabilities of software engineering personnel, and particularly their experience with the computer science issue and the applications domain issue of the project.
4. The environment, which is made up of the tools and techniques available to support efficient software development and to automate the process.
5. The required quality of the product, including its features, performance, reliability, and adaptability.

The required levels of the quality and personnel are assumed to be constant. The ordınate of the graph refers to software unit costs per SLOC, per function point, per component realized by the organization.

2.3 Pragmatic Software Cost Estimation

Software industry has not yet standardized the unit of measure for software projects. It is hard enough to collect a homogenous set of data within one organization, and it is extremely difficult to homogenize data across different organization with different processes, language, domains and so on.

Hence we still facing the following problems:

1. Which cost model to use?
2. How to measure software size in source lines of code or function points?
3. What constitutes a good estimate

Attributes of good software estimates

- It is conceived and supported by the project manager, architecture team development team, and test team accountable for performing the work.
- It is accepted by all stakeholders as ambitious but realizable.
- It is based on a well-defined software cost model with a credıble basis.

- It is based on a database of relevant project experience that includes similar processes, similar technologies, similar environments, similar quality requirements and similar people.
- It is defined in enough detail so that its key risk areas are understood and the probability of success is objectively assessed.

Boehm (1981) [3] discusses seven techniques of software cost estimation:

(1) Algorithmic cost modeling

A model is developed using historical cost information, which relates some software metric (usually its size) to the project cost. An estimate is made of that metric and the model predicts the effort required.

(2) Expert judgement

One or more experts on the software development techniques to be used and on the application domain are consulted. They each estimate the project cost and the final cost estimate is arrived at by consensus.

(3) Estimation by analogy

This technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects.

(4) Parkinson's Law

Parkinson's Law states that work expands to fill the time available. In software costing, this means that the cost is determined by available resources rather than by objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months.

(5) Pricing to win

The software cost is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customer's budget and not on the software functionality.

(6) Top- down estimation

A cost estimate is established by considering the overall functionality of the product and how that functionality is provided by interacting sub-functions. Cost estimates are made on the basis of the logical function rather than the components implementing that function.

(7) Bottom- up estimation

The cost of each component is estimated. All these costs are added to produce a final cost estimate.

2.4 Effort Estimation Techniques

Estimate Defined

According to the dictionary the verb 'estimate' means "To judge tentatively or approximately the value, worth, or significance of". Estimate implies a judgment, considered or casual, that precedes or takes the place of actual measuring or counting or testing out.

This definition immediately points out a risk. It states that estimators are approximating or acting tentatively. Project managers take the approximate number that scientific research provides and considering it fact or gospel. They take the estimate from the available algorithmic or analogy based models and turn it into an "expectation". One of our challenges during this work has been to manage that "expectation" and provide a method to systematically approach towards a solution to the problem.

Effort estimation is the most common technique for costing any engineering development project. A number of person-days, months, or years is applied to the solution of each project task. Cost is associated with each unit of effort and an estimated cost is derived. Like the LOC or FP technique, effort estimation begins with a delineation of software functions obtained from the project scope. A series of software engineering tasks-requirement analysis, design, code, and test – must be performed for each function.

The planner estimates the effort (e.g., person-months) that will require accomplishing each software engineering task for each software function. It is very likely that the labor rate will vary for each task. Senior staff is heavily involved in requirements analysis and early design tasks, code, and early testing.

Costs and effort for each function and software engineering task is computed as the last step. If effort estimation is calculated independently of LOC or FP estimation, we now have two estimates for cost and effort that may be compared and reconciled. If both sets of estimates show reasonable agreement, there is good reason to believe that the estimates are reliable. If, on the other hand, the results of these decomposition techniques show little agreement, further investigation and analysis must be conducted.

Estimation Models

An estimation model for computer software uses empirically derived formulae to predict data that are a required part of the software project-planning step. The empirical data that support most models are derived from a limited sample of projects. For this reason no estimation model is appropriate for all classes of software and in all development environments. Therefore, the results obtained from such models must be used judiciously.

Resource models consists of one or more empirically derived equations that predict effort (in person-months), project duration (in chronological months), or other pertinent project data. Basili [BAS80] describes four classes of resource models: static single-variable model, dynamic multivariable models, dynamic multivariable models, and theoretical models.

Static Single-Variable Models

The static single-variable model takes the form:

$$\text{Resource} = c_1 * (\text{estimated characteristic})^{c2}$$

Where the resource could be effort (E), project duration (D), staff sizes (S), or requisite lines of software documentation (DOC). The constant c1 and c2 are derived from data collected from past projects. The estimated characteristic is lines of source code, effort (if estimated), or other software characteristics. The basic COCOMO estimation model is an example of a static single-variable model.

Static Multivariable Models

Like their single-variable counterpart, this model makes use of historical data to derive empirical relationships typical model of this category takes the form:

$$\text{Resource}=c_{11}*e_1 + c_{21}*e_2+\ldots$$

Where e_i is the ith software characteristic and c_{i1},c_{i2} are empirically derived constants for the ith characteristic.

Dynamic Multivariable Models

This model projects resource requirements as a function of time. If the model is derived empirically, resources are defined in a series of time steps that allocate some percentage of effort (or other resource) to each step in the software engineering process. Each step may be further subdivided into tasks. A theoretical approach to dynamic multivariable modeling hypothesizes a continuous " resource expenditure curve", and from it, derives equations that model the behavior of the resource. The Putnam Model is a theoretical dynamic multivariable model.

For large projects, several cost estimation techniques should be used in parallel and their results compared. If these predict radically different costs, more information should be sought and the costing process repeated. The process should continue until the estimates converge.

Cost models are based on the fact that a firm set of requirements has been drawn up and costing is carried out using these requirements as a basis. However, sometimes the requirements may be changed so that a fixed cost is not exceeded.

Algorithmic Cost Modeling

Costs are analyzed using mathematical formulae linking costs with metrics.

The most commonly used metric for cost estimation is the number of lines of source code in the finished system (which of course is not known).

Size estimation may involve estimation by analogy with other projects, estimation by ranking the sizes of system components and using a known reference component to estimate the component size or may simply be a question of engineering judgement.

Code size estimates are uncertain because they depend on hardware and software choices, use of a commercial database management system etc.

An alternative to using code size as the estimated product attribute is the use of `function-points', which are related to the functionality of the software rather than to its size.

Function points are computed by counting the following software characteristics:

- External inputs and outputs.
- User interactions.
- External interfaces.
- Files used by the system.

Each of these is then individually assessed for complexity and given a weighting value which varies from 3 (for simple external inputs) to 15 (for complex internal files).

The function point count is computed by multiplying each raw count by the estimated weight and summing all values, then multiplied by the project complexity factors which consider the overall complexity of the project according to a range of factors such as the degree of distributed processing, the amount of reuse, the performance, and so on.

Function point counts can be used in conjunction with lines of code estimation techniques The number of function points is used to estimate the final code size. Based on historical data analysis, the average number of lines of code in a particular language required to implement a function point can be estimated (AVC). The estimated code size for a new application is computed as follows:

Code size = AVC x Number of function points

The advantage of this approach is that the number of function points can often be estimated from the requirements specification so an early code size prediction can be made.

Mathematical Estimation Models

The Rayleigh-Putnam Curve uses a negative exponential curve as an indicator of cumulative staff-power distribution over time during a project.

Technology constant, C, combines the effect of using tools, languages, methodology, quality assurance procedures. standards etc. It is determined on the basis of historical data (past projects). C is determined from project size, area under effort curve, and project duration.

Rating: C = 2000 -- poor, C = 8000 -- good, C = 11000 it is excellent.

Regression Models

COCOMO

Most widely used model for effort and cost estimation. It considers a wide variety of factors.

Projects fall into three categories: organic, semidetached, and embedded, characterized by their size

Cost Drivers for the COCOMO Model are :

- Software reliability
- Size of application database
- Complexity
- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language expertise
- Performance requirements
- Memory constraints
- Volatility of virtual machine
- Environment
- Turnaround time
- Use of software tools
- Application of software engineering methods
- Required development schedule
- Values are assigned by the manager.

Automated Estimation Tools

Automated estimation tools allow the planner to estimate cost and effort and to perform "what if" analyses for important project variables such as delivery date or staffing. All have the same general characteristics and require:

1. A quantitative estimate of project size (e.g., LOC) or functionality (function point data)
2. Qualitative project characteristics such as complexity, required reliability, or business criticality
3. Some description of the development staff and/or development environment

From these data, the model implemented by the automated estimation tool provides estimates of the effort required to complete the project, costs, staff loading, and, in some cases, development schedule and associated risk.

BYL (Before You Leap) developed by the Gordon Group, WICOMO (Wang Institute Cost Model) developed at the Wang Institute, and DECPlan developed by Digital Equipment Corporation are automated estimation tools that are based on COCOMO. Each of the tools requires the user to provide preliminary LOC estimates.

These estimates are categorized by programming language and type (i.e., adapted code, reused code, new code). The user also specifies values for the cost driver attributes. Each of the tools produces estimated elapsed project duration (in months), effort in staff-months, average staffing per month, average productivity in LOC/pm, and cost per month. This data can be developed for each phase in the software engineering process individually or for the entire project.

SLIM is an automated costing system based on the Rayleigh-Putnam Model. SLIM applies the Putnam software model, linear programming, statistical simulation, and program evaluation and review technique, or PERT (a scheduling method) techniques to derive software project estimates

The system enables a software planner to perform the following functions in an interactive session:

(1) calibrate the local software development environment by interpreting historical data supplied by the planner;

(2) create an information model of the software to be developed by eliciting basic software characteristics, personal attributes, and environmental considerations; and

(3) conduct software sizing--the approach used in SLIM is a more sophisticated, automated version of the LOC costing technique.

Once software size (i.e., LOC for each software function) has been established, SLIM computes size deviation (an indication of estimation uncertainty), a sensitivity profile that indicates potential deviation of cost and effort, and a consistency check with data collected for software systems of similar size.

The planner can invoke a linear programming analysis that considers development constraints on both cost and effort, and provides a month-by-month distribution of effort, and a consistency check with data collected for software systems of similar size.

ESTIMACS is a "macro- estimation model" that uses a function point estimation method enhanced to accommodate a variety of project and personnel factors.

The ESTIMACS tool contains a set of models that enable the planner to estimate

1. system development effort,
2. staff and cost,
3. hardware configuration,
4. risk,
5. the effects of "development portfolio."

The system development effort model combines data about the user, the developer, the project geography (i.e., the proximity of developer and customer), and the number of "major business functions" to be implemented with information domain data required for function point computation, the application complexity, performance, and reliability.

ESTIMACS can develop staffing and costs using a life cycle data base to provide work distribution and deployment information.

The target hardware configuration is sized (i.e., processor power and storage capacity are estimated) using answers to a series of questions that help the planner evaluate transaction volume, windows of application, and other data.

The level of risk associated with the successful implementation of the proposed system is determined based on responses to a questionnaire that examines project factors such as size, structure, and technology.

SPQR/20, developed by Software Productivity Research, Inc. has the user complete a simple set of multiple choice questions that address:

- project type (e.g., new program, maintenance),
- project scope (e.g., prototype, reusable module),
- goals (e.g., minimum duration, highest quality),
- project class (e.g., personal program, product),
- application type (e.g., batch, expert system),
- novelty (e.g., repeat of a previous application),
- office facilities (e.g., open office environment, crowded bullpen),
- program requirements (e.g., clear, hazy),
- design requirements (e.g., informal design with no automation),
- user documentation (e.g., informal, formal),
- response time,
- staff experience,
- percent source code reuse,
- programming language,
- logical complexity of algorithms,
- code,
- data complexity,
- project related cost data (e.g., length of work week, average salary).

In addition to output data described for other tools, SPQR/20 estimates:

- total pages of project documentation,
- total defects potential for the project,
- cumulative defect removal efficiency,
- total defects at delivery, and
- number of defects per KLOC.

Each of the automated estimating tools conducts a dialog with the planner, obtaining appropriate project and supporting information and producing both tabular and (in some cases) graphical output. All these tools have been implemented on personal computers or engineering workstations. Martin compared these tools by applying each to the same project.

A large variation in estimated results was encountered, and the predicted values sometimes were significantly different from actual values.
This reinforces the fact that the output of estimation tools should be used as one "data point" from which estimates are derived--not as the only source for an estimate.

The How of Estimating

Estimating processes can be classified as either model-based or analogy-based (Shepperd 1996). Analogy-based processes are sub-divided into formal and informal classifications. That leaves us with three approaches to estimating, defined below:

- Model-based – this approach, also known as algorithmic, uses a formal model such as COCOMO to estimate a software development project.

- Formal analogy-based – this approach, also known as proxy-based, uses a database of previous projects. New projects are estimated based on a formal and rigorous comparison with previous projects.

- Informal analogy-based – this approach, also known as rule-of-thumb, is based on the experience of the estimator who arrives at an estimate by informally comparing this project with previous projects and applies a number of rules-of-thumb

According to our survey [source: STQC, Director, Ministry of IT, India], the informal analogy-based process is the most commonly used estimating method. How well do one do using this approach? Most projects overshoot their estimated schedules by anywhere from 25 to 100 percent. The causes of these inaccurate estimates are many:

- Requirements creep is a major reason for overruns.
- Estimates are not based on past performance.
- In some cases, the estimator is being asked to establish goals for performance or accept previously established goals.
- Management, users, and clients often do not accept realistic estimates.

Estimates that are too low result in planning inefficiencies that drive up the actual cost of the project, not to mention loss of credibility.

Improving Estimates

A process for estimating includes the following four steps:

- Estimate the size of the development product.
- Estimate the effort (called Work in Microsoft Project) in effort-months.
- Estimate the schedule in calendar months.
- When the project finishes, collect actual data for use in future estimates.

The size of the development product or application can be estimated in many different ways. Lines-of-code, function points, and GUI components are examples of ways to measure the application. Each has advantages and disadvantages.

- Informal analogy-based – this approach, also known as rule-of-thumb, is based on the experience of the estimator who arrives at an estimate by informally comparing this project with previous projects and applies a number of rules-of-thumb.

According to our survey [source: STQC, Director, Ministry of IT, India], the informal analogy-based process is the most commonly used estimating method. How well do one do using this approach? Most projects overshoot their estimated schedules by anywhere from 25 to 100 percent. The causes of these ınaccurate estimates are many:

- Requirements creep is a major reason for overruns.
- Estimates are not based on past performance.
- In some cases, the estimator is being asked to establish goals for performance or accept previously established goals.
- Management, users, and clients often do not accept realistic estimates.

Estimates that are too low result in planning inefficiencies that drive up the actual cost of the project, not to mention loss of credibility.

Improving Estimates

A process for estimating includes the following four steps:

- Estimate the size of the development product.
- Estimate the effort (called Work in Microsoft Project) in effort-months.
- Estimate the schedule in calendar months.
- When the project finishes, collect actual data for use in future estimates.

The size of the development product or application can be estimated in many different ways. Lines-of-code, function points, and GUI components are examples of ways to measure the application. Each has advantages and disadvantages.

After you estimate the size of the application, use that number to estimate effort. Historical data for similar, completed project can be most helpful at this point. This is also the time to start managing project stakeholders' expectations. Even if one come up with a point estimate for the effort, present it as a range or use confidence factors. One can use the table (McConnell 169) below to determine the range.

	Effort and Size	
Phase	Optimistic	Pessimistic
Initial product concept	0.25	4.0
Approved product concept	0.50	2.0
Requirements specification	0.67	1.5
Product design specification	0.80	1.25
Detailed design specification	0.90	1.10

Table 2.1: Estimate Multipliers by Project Phase

The 'optimistic' column can be used for the 5% confidence factor and the 'pessimistic' column can be used for the 95% confidence factor. Now that we have estimated the effort of our project, we can determine the schedule. Unless you have a better method, you can use the following rule-of-thumb (McConnell 183) to translate your effort estimate into a schedule estimate:

$$\text{Schedule in months} = 3.0 \text{ x effort-months}^{1/3}$$

Here are some other ideas one might want to apply to the estimating process:

- Use documented data from organization's own similar past projects.
- Collect and analyze historical data including:

1 Basic characteristics of the development process.

2. All estimates and re-estimates.

3. Product characteristics including size, complexity, description, and classification of software.

- Use several estimators and several techniques and compare results. Formalize when and how cost estimates are performed (Vigder 4).
- Re-estimate throughout the lifecycle.

Now that one has estimated the size, effort, and schedule of the project, completed the development, and installed the application into production, recording the actual results so that this hard-earned information to may be used to improve the estimate on the next project.

Estimation by Analogy

Martin Shepperd and Chris Schofield [17] gave this method of estimation. This is an approach to estimation based upon the use of analogies. The underlying principle is to characterize projects in terms of features (for example, the number of interfaces, the development method or the size of the functional requirements document). Completed projects are stored and then the problem becomes one of finding the most similar projects to the one for which a prediction is required. Similarity is defined as Euclidean distance in n-dimensional space where n is the number of project features. Each dimension is standardized so that all dimensions have equal weight. The known effort values of the nearest neighbors to the new project are then used as the basis for the prediction. The process is automated using a PC-based tool known as ANGEL. The method is validated on nine different industrial datasets (a total of 275 projects) and in all cases analogy outperforms algorithmic models based upon stepwise regression. From this work, Shepperd and Chris argue that estimation by analogy is a viable technique that, at the very least, can be used by project managers to complement current estimation techniques.

Originally, analogy estimates were created by a very labor intensive manual search. It quickly became obvious, however, that the process would need to be automated if it were to be used seriously. To this end ANGEL (ANaloGy softwarE tooL) was developed in Visual Basic. ANGEL is an environment where data can be stored, analogies found and estimates generated.

A technique known as jack-knifing is employed by ANGEL when assessing the level of confidence that can be placed in the predictions from a given dataset. This involves successively removing each project from the database and using the remaining projects as the analogy source. The Mean Magnitude of Relative Error (MMRE) is then found by computing the mean of all the percentage errors which result from comparing the estimate with the actual effort figure. The MMRE figure gives an indication of the likely accuracy that can be expected when estimating future projects. It is also useful for technique comparison because, unlike $R^{2,}$ it is estimation method independent.

The analogy approach offers a number of advantages. First the approach is simple and easy to understand (unlike say neural nets). Estimation by analogy has a lot in common with expert judgement. This leads to the other advantage of the analogy approach over other techniques in that it can be regarded as a "glass box", i.e. it is possible to directly relate output to the inputs. This is desirable as it allows the user to judge the quality of the estimate by viewing its source. Next, the method is more robust in that it can cope with outliers and datasets for which no statistical relationships can be found. Last, but not the least, it has proved to be more accurate than the traditional algorithmic methods for several datasets.

The calibration of the analogy-based method requires the detection of the best configuration of the available method options. The options that may be adjusted are (a) the distance metric by which the projects of the database will be sorted according to their similarity to the one under estimation (e. g. Euclidean distance, Manhattan distance),(b) the number of closest projects (analogies), (c) the set of attributes for judging analogy, and (d) the statistic that will be computed from the efforts of the closest projects and will serve as estimation for the unknown effort. In addition, the statistic may be size-adjusted.

The bootstrap method, BRACE

The method of analogies, as implemented in the ANGEL tool [17], foresees a method-tuning phase focusing on the search of the appropriate project attributes for the estimation procedure. The method also results in point estimation for the unknown effort, i.e. a single value, which is computed from a particular sample, the historical data set coming from a practically infinite population with unknown characteristics of all possible software projects. A critical question is how valid the tuning activity may be and which is the accuracy and reliability of the generated estimation. In these cases, it is common practice to enhance the value of the point estimation with various measures of accuracy from its probability distribution. assumption on the population distribution. The second method is based on the assumption that the dataset comes from a theoretical distribution, which is approximated by a parametric model such as the normal distribution. The non-parametric bootstrap method is applied when we use a random sample of size n to estimate a certain parameter (in our case the effort of a new project) by a statistic which is calculated independently from the order of the sample. The idea is to draw a large number (say B) of samples from the original sample with replacement and to calculate the statistic for each one of these samples. These integers determine which members of the original sample are selected to participate in the new bootstrap sample. An obvious consequence of this sampling method is that in every new bootstrap sample there are data points, which appear more than once while others, do not appear at all.

The parametric bootstrap differs in the way of sampling. Instead of sampling with replacement from the original data points, we draw B samples of size n (same size as in the original data) from a theoretical distribution that is assumed to have generated the data.

The authors have implemented a software tool, called BRACE (BootstRap based Analogy Cost Estimation), that supports the practical application of the analogy based method using the bootstrap sampling method described above. In the first version of BRACE they have implemented only non-parametric bootstrap. Bootstrap is used for two purposes: first in order to calibrate the method of analogies choosing the optimal options (distance metric, number of analogies, statistic, and size adjustment) and second to assess the accuracy of the estimations obtained by the method. The tool provides a flexible interface that allows the user to experiment with the different calibration options. In the calibration phase, the tool works on a data set of historical projects. It estimates each project's effort in turn using the other projects to find analogies (jack-knife).

Accuracy is determined in terms of the MMRE and Pred(25) criteria. After the calibration phase the tool may estimate new projects using the method configuration that was found to be the best.

Chapter 3
The Proposed Model

3.1 Estimation by Analogy

The proposed model is based on the concept of estimation by analogy (given by M.J. Shepperd, 1997) [17]. We have enhanced the original model given by Shepperd in two different ways. Our enhancement is in the terms of the number of project attributes that characterize a project. We have also tested various methods to deal with the sparse data ,through various versions of the model. While scaling the factors, we have incorporated practical working environment prevailing in some of the CMM optimizing level organizations which we surveyed during the course of this work. The model is constructed to serve the purpose of both estimating the effort for the new project and entering data of the completed project in the data file, for its usage in the future estimations.

3.2 Factors Considered

We worked on 1238 projects from the ISBSG dataset. While comparing the new project with the older ones, our model makes sure that only projects that measure size in the same way as done in the new project is considered. Our model compares the new project with other projects, which have used the same sizing method, (e.g. IFPUG, Mark II, NESMA, COSMIC-FFP etc).

We grouped the factors available in the dataset in three different categories namely Category-I, Category-II and category-III , depending on the information which could be extracted from them.

Category-I

Factors in this category were used to characterize the project for assessing similarity of projects by Euclidean distance calculation The following factors were included in this category:

Function Points

This field describes the adjusted function point count number , adjusted by the **Value Adjustment Factor** which is the adjustment to the function points, applied by the project submitter, that takes into account various technical and quality characteristics e.g.: data communications, end user efficiency etc. This data is not reported for some projects in the data set (i.e. it equals 1 in such cases).

Development Type

This field describes whether the development was a new development or enhancement. We proposed the corresponding cost driver values in COCOMO II as shown in Table 3.1.

Factor/ COCOMO II Factor	AEXP	ACAP	VEXP	LEXP	PCAP	FINAL
Enhancement	Nominal(1.0)	High(.86)	High(.86)	High(.9)	High(.86)	.632
New Development	High(.91)	Very High(.71)	Nominal(1.0)	Nominal(1.0)	High(.86)	.55

*Table 3.1: The corresponding cost driver values in COCOMO II for Development Type

Development Platform

Defines the primary development platform, as determined by the operating system used. Each project is classified as either, a PC, Mid Range or Mainframe. We selected the corresponding cost driver values in COCOMO II as shown in the Table 3.2.

Factor/COCOMO II Factor	TIME	STOR	VIRT	TURN	FINAL
Main frame	Nominal(1.0)	Nominal(1.0)	Nominal(1.0)	Nominal(1.0)	1.0
Mid Range	High(1.11)	High(1.06)	High(1.15)	High(1.07)	1.32
PCs	Very High(1.3)	Very High(1.21)	Very High(1.3)	Nominal(1.0)	2.0

*Table 3.2: The corresponding cost driver values in COCOMO II for Development Platform.

Application Type

This identifies the type of application being addressed by the project. (e.g.: information system, transaction/production system, process control.) We divided them in four levels according to the required reliability, product complexity and database size. We used COCOMO II procedure to form these levels.

Level 1	=	Transaction and Production System
Level 2	=	Control and Real Time System
Level 3	=	Management and Office Information System
Level 4	=	Others

We took the COCOMO II factors corresponding to these as shown in the Table 3.3.

Factor/COCOMO II Factor	RELY	CPLX	DATABASE	FINAL
Level 1	Very High(1.40)	Nominal(1.0)	Very High(1.16)	1.62
Level 2	Very High(1.40)	High(1.15)	Nominal(1.0)	1.61
Level 3	High(1.15)	Nominal(1.0)	High(1.08)	1.24
Level 4	Nominal(1.0)	Nominal(1.0)	Nominal(1.0)	1.0

*Table 3.3: The corresponding cost driver values in COCOMO II for Application Type

* Though Table 3.1, 3.2 and 3 3 are not of use when estimating by analogy, but the proposed rating might be useful when using COCOMO II to confirm on the estimates given by analogy model.

Modern Programming Practices

We have taken two factors to gather information regarding modern programming practices. They are namely Language type and Primary Programming Language.

Language Type

Defines the language type used for the project, e.g. 3GL, 4GL, Application Generator etc.

Primary Programming Language

The primary programming language used for the developments are JAVA, C++, PL/1, Natural, Cobol etc.

We have done the classification of these languages in three classes as follows:

CLASS1->[ORACLE, SQL, Pl-SQL, ACCESS, SAP ABAP]

CLASS2->[JAVA,VB, C++]

CLASS3->[COBOL V2, Pro*C, Lex]

The classification is done according to the information gathered form a Bangalore based CMM level 5 company.

The views of the programmers in this company (in their words) are listed below:

1. Oracle is used in the form of builders and report writers, this language provide quite beautiful libraries and error handling. In terms of database it is the fastest and widely accepted as the only choice in case there are no constraints from the implementation side. Most operating systems are slower than oracle database. So one can count on oracle as very fast and reliable language. Still there are a lot of bugs

and modifications are going on. It provides reliable solutions. Rapid application development is possible if specifications are proper.

2. In JAVA, wide ranges of API's are available on the net. So one doesn't have to generate all the code with his or her effort. This is the primary advantage. The disadvantage is that without the use of pointers the application is bound to be slow. Rapid application is possible if one is doing something that has already been done and code is available in hand. Encapsulation provides modification to any module.

3. C++ is quite fast and with object oriented approach one can develop any application on this language. Pointers are allowed, so one does not loose time in searching the classes. Developments of one's own code is easy. Here also all the benefits of Java are available and one can have pointers as well. but it lacks in web technologies. So choice is not there in case of web applications one have to go to Java only.

4. Anything one can think can be done in SQL with less effort. The programmers find it very interesting.

5. In VB, visual attributes are very good and reliable.

6. Pro*C is the combination of C and SQL. On SQL front, it is under development. The effort required is fairly more. Also, not many have good knowledge of it.

7. COBOL V2 takes time to develop applications.

8. Lex is the fastest in terms of code generation at run time, but in most cases much effort is desired.

9. Pl-SQL it is becoming faster and code generation is also becoming faster as PL/SQL engine is upgraded. It is still under development and new releases will be there soon which will make life lot easier. It is a fourth generation language, one generation ahead of languages like java, c++. It is highly object oriented.

We categorized the experiences of the software practitioners in Table 3.4.

	A	**B**	**C**
CLASS 1	Less	High	High
CLASS 2	Moderate	High	High
CLASS 3	High	Low	Low

Table 3.4: Classification of Primary Programming Languages

Where A stand for the effort required for generating specified lines of code, B stand for the easily available modules, which can be modified for the new project, and C stand for the prior knowledge and experience of the programmer in that language.

DBMS Used

GROUP1-> [DBMS NOT USED]

GROUP2-> [ORACLE, SQL Server, ACCESS]

GROUP3-> [OTHERS]

Group2 lists the familiar database languages in which the working team has enough training and experience. Group3 includes those languages, which are very rarely used, and thus may require extra effort for learning and implementing.

Implementation Date

This is the actual date of implementation of the completed project.

Category-II

Factors in this category were used to provide the user with the additional information that is useful while comparing the new project attributes with the nearest and the next nearest analogy obtained from the model. The following factors were included in this category:

Function Size Metric Used

This describes the functional size metric used to record the size of the project, (e.g.. IFPUG3, IFPUG4, In-house etc.).

Reference Table Approach

This describes the approach used to handle counting of tables of code or reference data, (a comment field).

Data Quality Rating

This field contains an ISBSG rating code of A, B, or C applied to the project data by the ISBSG quality reviewers to denote the following:

A= The submission satisfies all the criteria for seemingly sound data.

B= The submission appears fundamentally sound but there is some evidence to question some of the supplied data.

C= The submission has some fundamental shortcomings in the data

Maximum Team Size

This describes the maximum number of people that worked at any time on the project, (peak team size).

How Methodology Acquired

This describes whether the methodology was purchased or developed in-house.

User Base - Concurrent Users

Number of users using the system concurrently.

Development Techniques

This describes the techniques used during development. (e.g.: JAD, Data Modeling, OO Analysis etc.).

Organization Type

This identifies the type of organization that submitted the project. (e.g.: Banking, Manufacturing, Retail).

Resource level

The data collected was recorded at the four-resource level in the data set, which is as follows

- 1 = development team effort (e.g., project team, project management, project administration)
- 2 = development team support (e.g., database administration, data administration, quality assurance, data security, standards support, audit & control, technical support)
- 3 = computer operations involvement (e.g., software support, hardware support, information center support, computer operators, network administration)
- 4 = end users or clients (e.g., user liaisons, user training time, application users and/or clients)

Category-III

We have also included the COCOMO cost drivers which are not mentioned in the ISBSG dataset. These factors included as input from the user for future project estimation. We could not include them in our model testing and result as no project in the dataset had these fields. These factors are as follows:

Product attributes refers to the constraints and requirements placed upon the project to be developed. These included

- ➢ Required software reliability (RELY)
- ➢ Database size (DATA)
- ➢ Product complexity (CPLX)

Computer attributes refer to the limitations placed upon development effort by the Hardware and operating system are being used to run the project. These limitations are listed below.

- ➢ Execution time constraint (TIME)
- ➢ Main storage constraint (STOR)

- Virtual machine volatility (VIRT)
- Computer turnaround time (TURN)

Personnel attributes refer to the level of skill that is possessed by the personnel. The Skills in question are general professional ability, programming ability, experience with the development environment and familiarity with the project's domain. These skills are characterized below.

- Analyst capabilities (ACAP)
- Applications experience (AEXP)
- Programmer capabilities (PCAP)
- Virtual machine experience (VEXP)
- Programming language experience (LEXP)

Project attributes refer to the constraints and conditions under which project development takes place. The issues that affect development are:

- Modern programming practices (MODP)
- Use of software tools (TOOL)
- Required development schedule (SCED)

3.3 Model Calibration

3.3.1 Scaling of Category-I Factors

We used Category-I factors as project characterizing attributes while calculating the Euclidean distances of the new project from the completed projects. The scaling of each factor is done on the scale of 0-10.This is essential to maintain uniformity while calculating Euclidean distances.

The following formula was used for scaling:

(Value of the Factor in the project concerned)/(Maximum value which that Factor can have)*10

The maximum value for the FP in the dataset we used was 20000.The maximum value of Implementation Year we took was 2002.

The value of a factor other then Function points and Implementation year was based on the level which they belong. The Table 3.4 summarizes the various levels for a factor. Thus the maximum value that could be assigned to these factors was 3. This value was then scaled on a scale of 0-10 before using them to calculate Euclidean distances.

Factors	**Level 1**	**Level 2**	**Level 3**	**Level 4**
Application Type	1	2	3	4
Development Type	New Development	Enhancement		
Development Platform	PC	MF	MR	
Language Type	3GL	4GL	APG	
Primary Programming Language	CLASS1	CLASS2	CLASS3	
DBMS Used	GROUP1	GROUP2	GROUP3	

Table 3.5: Factor Levels

3.3.2 Factor Adjustment

In our model we have taken Function Points as the metric for size of the product. Function Point analysis appears to have advantages over lines of code as a measure of software size for use in estimating software development cost, and there is widespread industry support for this method.

The adjustment factor which modifies the UFP count has the range of -35% to +35%. This means that if we have FP equal to 1, it may have its actual value anywhere from 0 65 to 1.35.

To adjust this possible error in the FP calculation we have followed the following calculation:

100% of FP contribution on the distance calculation can lead to the maximum of 135% distance calculated, therefore about 74% of FP contribution will lead to 100% of the distance calculated. Thus, we gave 74% weight to FP and 26% weight to the rest of the factors.

3.4 Euclidean Distance Calculation

These measure true straight-line distances in Euclidean space. Note that the distance from Kanpur to New Delhi would not be Euclidean unless you bored a tunnel through the earth. The flight path of plane would follow the earth's curvature and hence would not be a straight-line distance.

In a univariate example the Euclidean distance between two values is the arithmetic difference, i.e. $value_1$ - $value_2$. In the bivariate case the minimum distance is the hypotenuse of a triangle formed from the points. For three variables the hypotenuse will be extended through 3-dimensional space. Although difficult to visualize an extension of the Pythagorean theorem will give the distance between two points in *n*-dimensional space.

For our model the following formula has been used to evaluate the proximity of other projects in the data file from the new project:

Distance of NP from I^{th} CP =[0.74*(*((scaled level of NPF1-scaled level of CP(I)F1)^2+).26*((scaled level of NPF2-scaled level of CP(I)F2)^2 +(scaled level of NPF3-scaled level of CP(I)F3)^2)+......for all factors)]^0.5

Where NP is the New Project, CP is the completed project. I is the count for the serial number of the completed project in the data file and F1 is Function Points ,F2 onwards are other factors in Category-I.

Thus the distances of the new project from all the completed projects is calculated and the nearest two such projects are chosen. These two nearest project details along with there actual efforts are displayed to the user to make a decision regarding the new project.

3.5 Predicting with Sparse Data

The Version 1 of the model do not incorporate the missing data field completed projects. Thus in version one we have taken the *list-wise deletion approach.* With only 4% of the values missing at random in each of the factors, we lost 25% of the observations with list-wise deletion. In Version 2.1 and 2.2 we replaced the missing field with respectively the maximum and the minimum value of that field. The Version 3 of the model incorporates the missing data using the concept of *Hot-Deck Euclidean Distances Method.* This method involves filling in missing data by taking values from other observations in the same data set [18]. The choice of which value to take depends on the observation containing thc missing data. The latter property is what distinguishes the hot-deck imputation method from mean imputation. In addition to reducing non response bias and generating a complete dataset, hot-deck imputation preserves the distribution of the sample population. Unlike mean imputation, which distorts the distribution by repeating the mean value for all the missing observations, hot-deck imputation attempts to preserve the sample distribution by substituting different observed values for each missing observation. Hot-deck imputation selects an observation (donor) that best matches the observation containing the missing value (client). The donor then provides the value to be imputed into the client observation. Our model calculates Euclidean distances between each observation that contains missing data and all observations in the complete set.

3.6 Confidence Level

To measure the confidence level of the estimation made, we took the following relationship:

Confidence Level = (Maximum Square Distance possible - Square Distance of the analogous project from the new project)*100 / Maximum Square Distance possible

3.7 Model Limitation

Our Model has the following Limitations:

1. Using this method, we have to determine how best to describe projects. The choice of variables must be restricted to information that is available at the point that the prediction required. Possibilities include the type of application domain, the number of inputs, the number of distinct entities referenced, the number of screens and so forth.

2. Even once we have characterized the project, we have to determine the similarity and how much confidence can we place in the analogies. Too few analogies might lead to maverick projects being used; too many might lead to the dilution of the effect of the closest analogies. We have measured Euclidean distance in n-dimensional space where each dimension corresponds to a variable. Values are scaled so that each dimension contributes equal weight to the process of finding analogies. Generally speaking, two analogies are the most effective.

3. Finally, we have to derive an estimate for the new project by using known effort values from the analogous projects. Possibilities include means and weighted means which will give more influence to the closer analogies. Our model provides user with the nearest two project details, leaving on the user to take appropriate decision about the new project.

3.8 Data Set Management

3.8.1 Metrics Data Collection

To develop accurate estimates, a historical baseline must be established. The baseline consists of data from past software development projects. To be an effective aid in cost and effort estimations, baseline data must have the following attributes:

1. Data must be reasonably accurate – " guestimates " about past projects are to be avoided
2. Data should be collected for as many projects as possible
3. Measurements must be consistent (e.g. KLOC must be interpreted consistently across all projects for which data are collected)
4. Applications should be similar to work that is to be estimated-it makes little sense to use a baseline for batch information system work to estimate a real time microprocessor application

The process for establishing a baseline is illustrated in fig.3.3. Ideally, data needed to establish a baseline has been collected in an ongoing manner. Sadly, this is rarely the case. Therefore, data collection requires a historical investigation of past projects to reconstruct the required data. Once data has been collected (unquestionably the most difficult step), metrics computation is possible. Depending on the breath of data collected, metrics can span a broad range of KLOC or FP measures.

Finally, computed data must be evaluated and applied in instrumentation. Data evaluation focuses on the underlying reasons for the results obtained. Are the computed averages relevant to the project at hand? What extenuating circumstances invalidate certain data for use in this estimate? These and other questions must be addressed so that the metrics data are not used blindly.

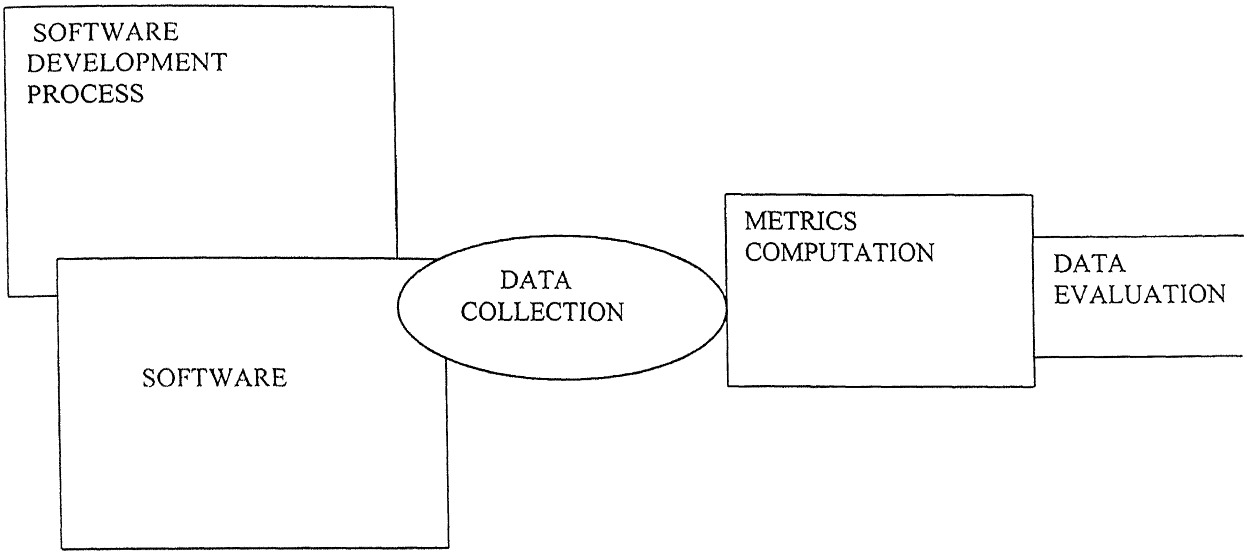


Fig 3.1: Process for Establishing a Baseline for Metrics Data Collection

3.8.2 Data Storage

The primary source of estimation by analogy is the Data set which is available in the organization. The recording data of all possible effort effecting factors of completed projects should be done to make successful decisions while undertaking any new project. Our model incorporates the function of storing the data of all factors of Category-I, Category-II and Category-III of completed project. Each added completed project will enhance the precision of effort estimation.

Chapter 4
Software Development

4.1 Software Requirements Specification (SRS)

To test the model we developed the software in Java whose SRS was documented. The SRS of the model software is given below:

1. Introduction

1.1 Purpose

The intent and purpose of the present document is to present the requirement specifications for the software product entitled "AN ENHANCED ANALOGY BASED MODEL FOR THE EFFORT ESTIMATION OF SOFTWARE PROJECTS". The language and the nature of this SRS document shall be concise and the technical knowledge of the reader shall not limit its comprehension.

1.2 Scope

This document is meant for the following audience.

- The Project Managers of software projects
- Software developer(s) who shall be involved with the project and developing extended versions of the same..
- Any other authorized person.

It covers the broad functionality, system, reliability and other relevant attributes required from the project. However, it does not describe any implementation details in so far that technical expertise may not be required to understand it

1.3 References

For the purpose of designing and developing this document, the following websites/technical-papers have been referred to.

(a) Writing Software Requirements Specifications [Donn Le Vie, Jr available at http://www.raycomm.com/techwhirl/ softwarerequirementspecs.html]

(b) An Integrated Approach to Software Engineering [Pankaj Jalote, 2nd edition, Narosa Publishing House]

2. Overall Description

2.1 Overview

Accurate software effort estimation is an important part of the software process. Originally, estimation was performed using only human expertise, but more recently attention has turned to a variety of algorithmic and analogy based methods. This model attempts to present the potential of analogy based estimation in software effort estimation , in terms of accuracy and ease of use.

2.2 Product Perspective

It is with this objective in mind that this model has been conceived. It is a potent tool in the hand of such Project Managers who would like to keep a record of their experiences of completed projects and their estimation as well.

2.3 Product Functions

This model has been designed to serve the following functions:

- To maintain a database of completed projects by a organization.
- Estimating New Project Features .
- Updating the Existing Records.
- To print two nearest analogous projects.

2.4 Users

This model is essentially a single-user system. This single user is a Project Manager in software projects. The following characteristics of the user are noteworthy.

- The user has been using the personal computer and is not new to it.
- The user has no formal knowledge of analogy based approach concepts.
- The user is assumed to be unfamiliar to JAVA as well and, in all probability, will have to be trained in its usage.
- The user is an experienced Project Manager and is engaged in estimation activities.

2.5 Operating Environment

- This model is intended to be used in a software organization or a research environment
- The software will be running on an average personal desktop computer that has a Win-95/98/00/NT OS.

2.6 Constraints

The system will be subject to the following constraints as far as its operating environment, end-users and functionalities are concerned.

3. Functional Requirements

3.1 Introduction

The overall functionality is similar to that of any normal file handling viz. maintaining a data file of completed projects. Here a more detailed account of the system functionalities are provided. The functionalities have been ordered according to their importance in the system.

3.2 Inputs

The inputs to the system will be of manually fed by the Project Manager. The inputs will be as follows:

New and Completed Project Data

As soon as a new project arrives or a project is completed the Project Manager performs his/her analysis personally. The data collected by the Project Manager is generally of the following nature.

Client Information

This shall include the unique ID assigned to the new project (in consonance with the organization /Project Manager norms), the project's ID, application type, business type etc.

Factor Values

This shall include the project's scaled values on various factors in Category-I, Category-II and Category-III. The missing data is left without filling the space assigned.

If its the new project then the Project Manager uses the model for searching the two nearest analogies.

3.2 Processing

The input data is processed by the model to arrive at the two nearest analogies (Refer chapter 3).

3.4 Output

The output will be information regarding the two nearest projects which are analogous to the new project. This information will consist of effort in staff hours, recording method, resource level, maximum team size, methodology acquired, development techniques, user base (concurrent users),organization type, data quality rating, value adjustment factor and the confidence level.

4. External Interface Requirements

4.1 User Interfaces

The user interface will be through GUI.

4.2 Hardware Interfaces

The software will be built using the JAVA application. Hence, the back-end interfacing with the OS (Win family) will not be a concern for the software.

4.3 Software Interfaces

The software must be so extensible that the input that is received from the user may have to be read from another application viz. MS-Excel TM. This is required because the user has been working previously on Excel-spreadsheets and the large quantity of research-oriented data might be blocked. On the other hand the Project Mangers may be asked to store the data file in the text format.

5. Performance Requirements

Due to the high scope of the software, the performance requirements are high. The maximum data file size should not be less than 50000 records.

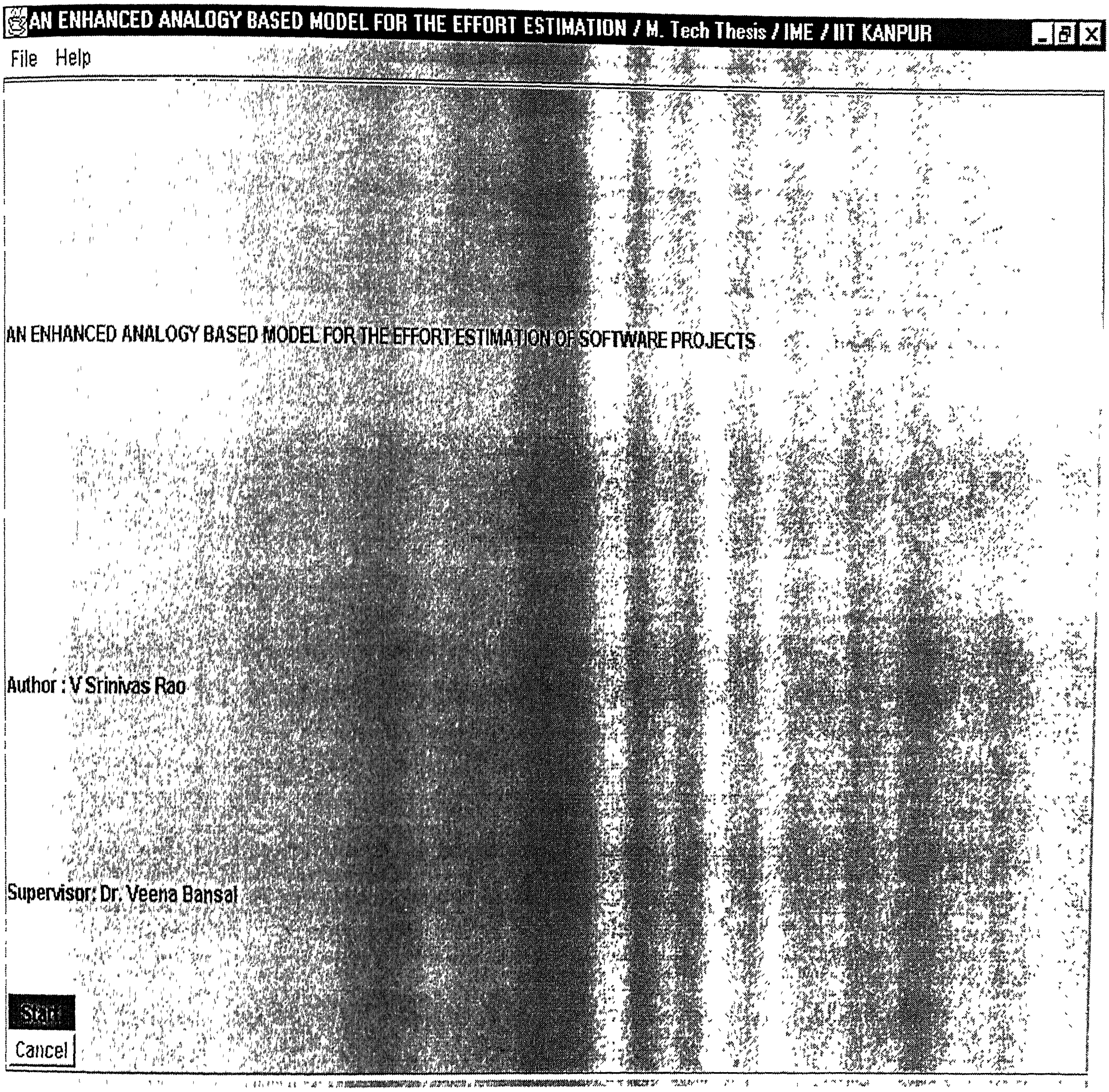
The speed at which the software is required to operate is nominal. A processing rate of 10-20 seconds per query is acceptable.

6. Design Constraints

The software is for personal use and is not subject to any standard design constraint. However, the international conventions with regard to security for various technical data must be adhered to. This information must be obtained after thorough consultation with the Project Managers and relevant documents.

4.2 A Sample Walk Through

The various user interfaces and output screens of the software developed are shown below.



AN ENHANCED ANALOGY BASED MODEL FOR THE EFFORT ESTIMATION / M. Tech Thesis / IME / IIT KANPUR

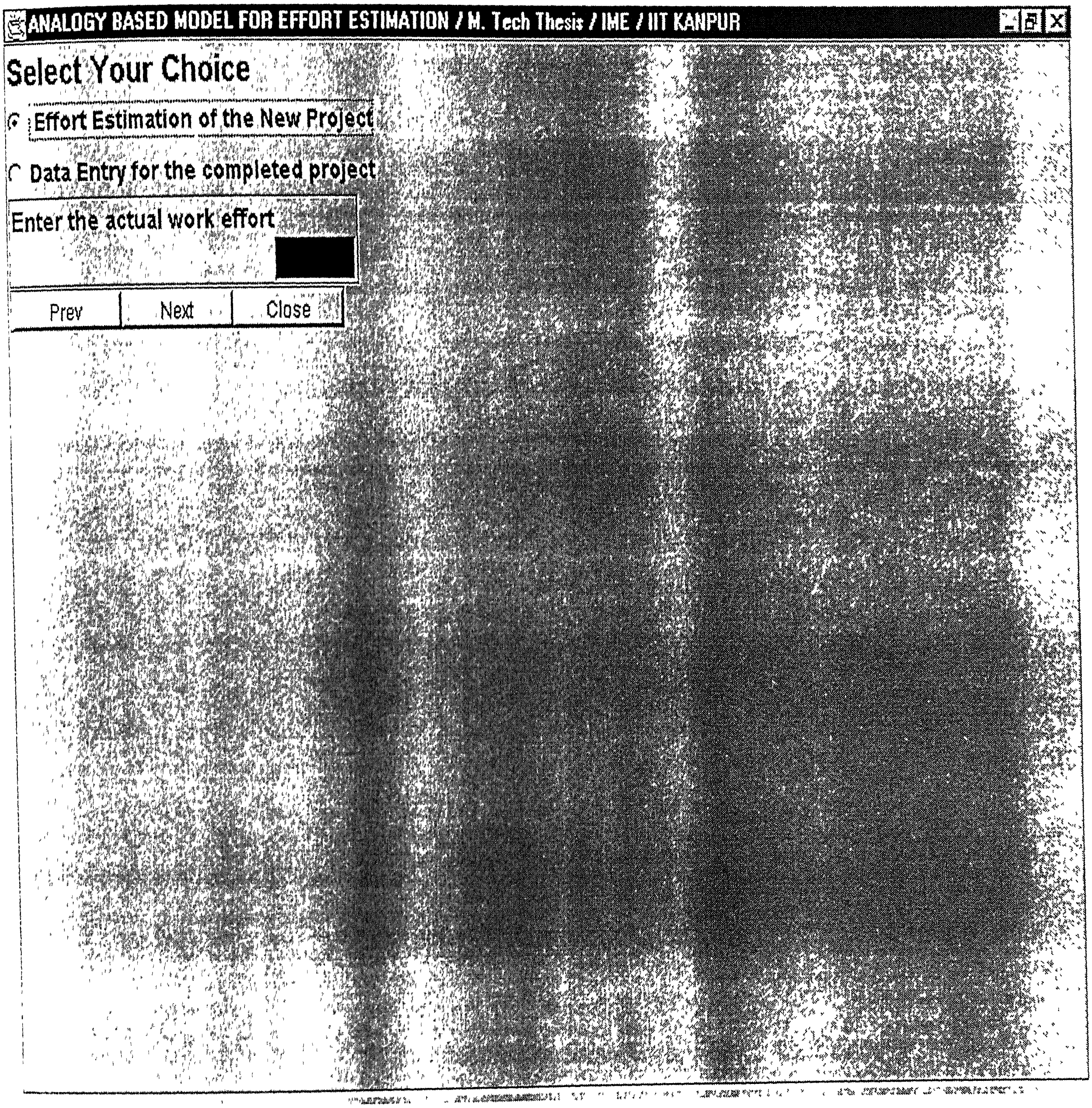
AN ENHANCED ANALOGY BASED MODEL FOR THE EFFORT ESTIMATION OF SOFTWARE PROJECTS

Author : V Srinivas Rao

Supervisor: Dr. Veena Bansal

Figure 4.1 Start-Up Screen

The other screen shown in fig. 4 2 registers the choice of the user for either effort estimation of the new project or data entry for the completed project. This screen also registers the actual work effort in the case of data entry for the completed projects.



ANALOGY BASED MODEL FOR EFFORT ESTIMATION / M. Tech Thesis / IME / IIT KANPUR

Select Your Choice

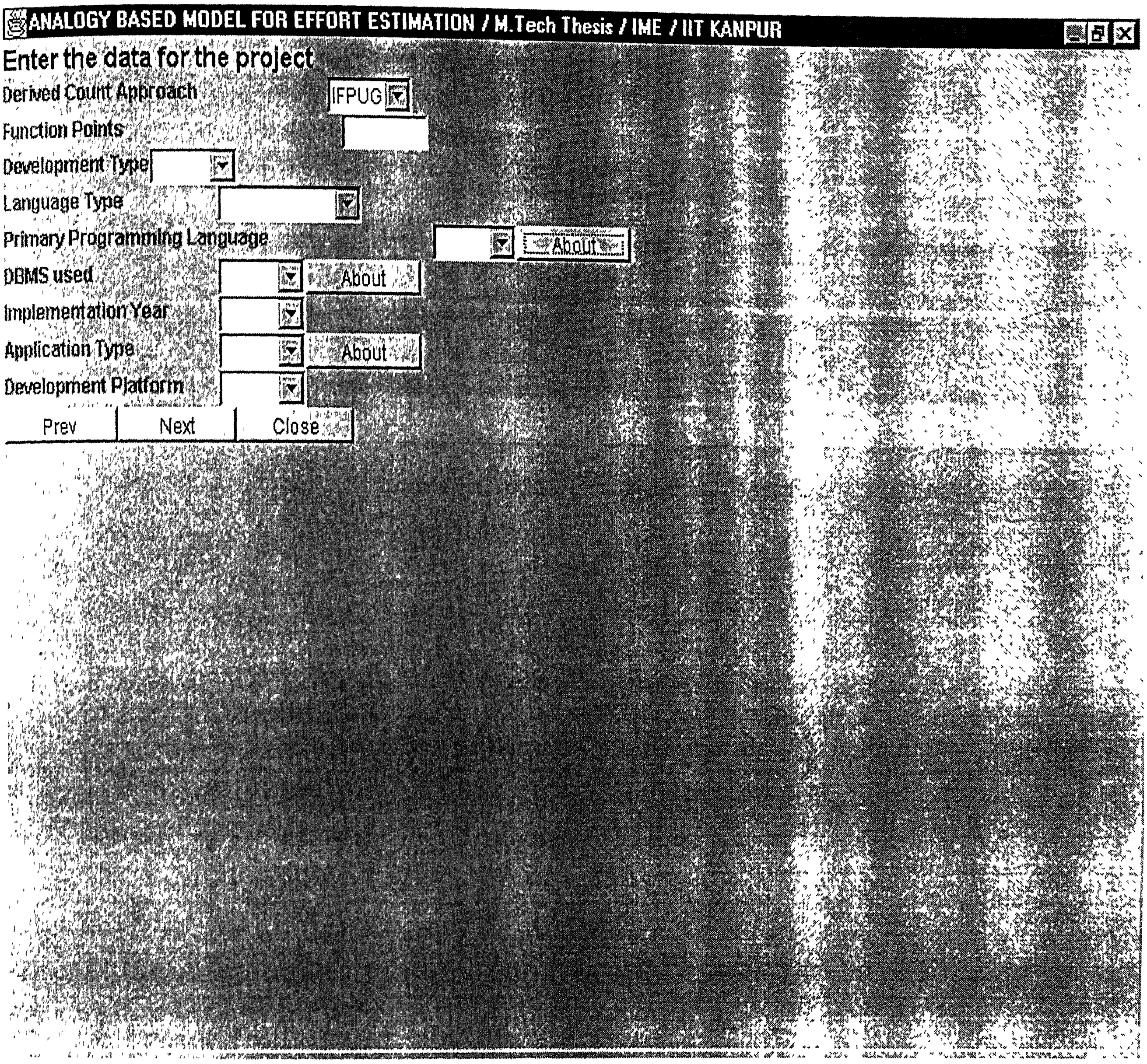
● Effort Estimation of the New Project

○ Data Entry for the completed project

Enter the actual work effort

Figure 4.2 Choice Menu

Fig. 4.3 and Fig. 4.4 enters the data for the project.



ANALOGY BASED MODEL FOR EFFORT ESTIMATION / M.Tech Thesis / IME / IIT KANPUR

Enter the data for the project

Derived Count Approach: IFPUG

Function Points

Development Type

Language Type

Primary Programming Language

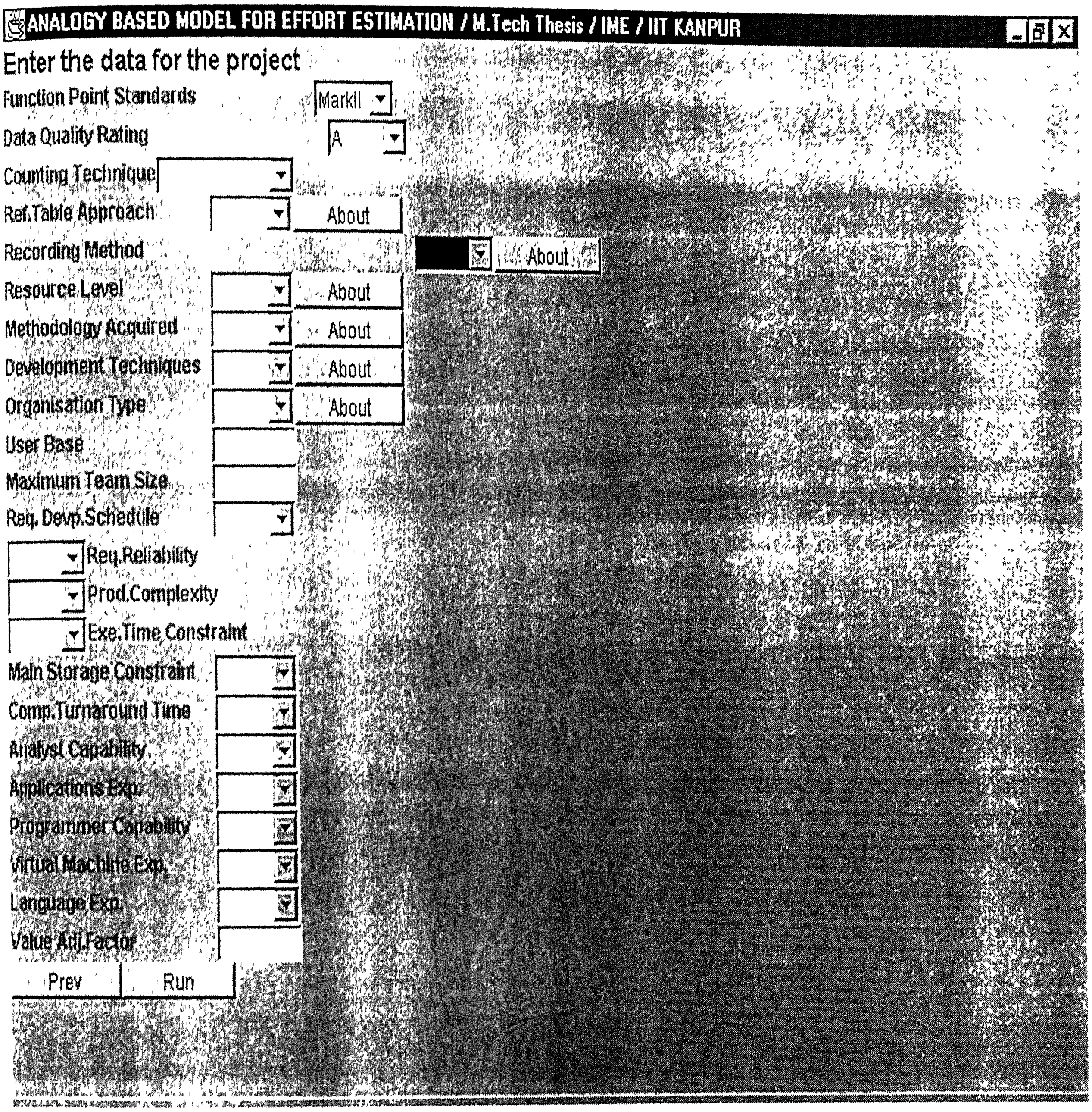
DBMS used

Implementation Year

Application Type

Development Platform

Figure 4.3 Factor Input Screen-1



ANALOGY BASED MODEL FOR EFFORT ESTIMATION / M.Tech Thesis / IME / IIT KANPUR

Enter the data for the project

Function Point Standards: MarkII

Data Quality Rating: A

Counting Technique

Ref.Table Approach

Recording Method

Resource Level

Methodology Acquired

Development Techniques

Organisation Type

User Base

Maximum Team Size

Req. Devp.Schedule

Req.Reliability

Prod.Complexity

Exe.Time Constraint

Main Storage Constraint

Comp.Turnaround Time

Analyst Capability

Applications Exp.

Programmer Capability

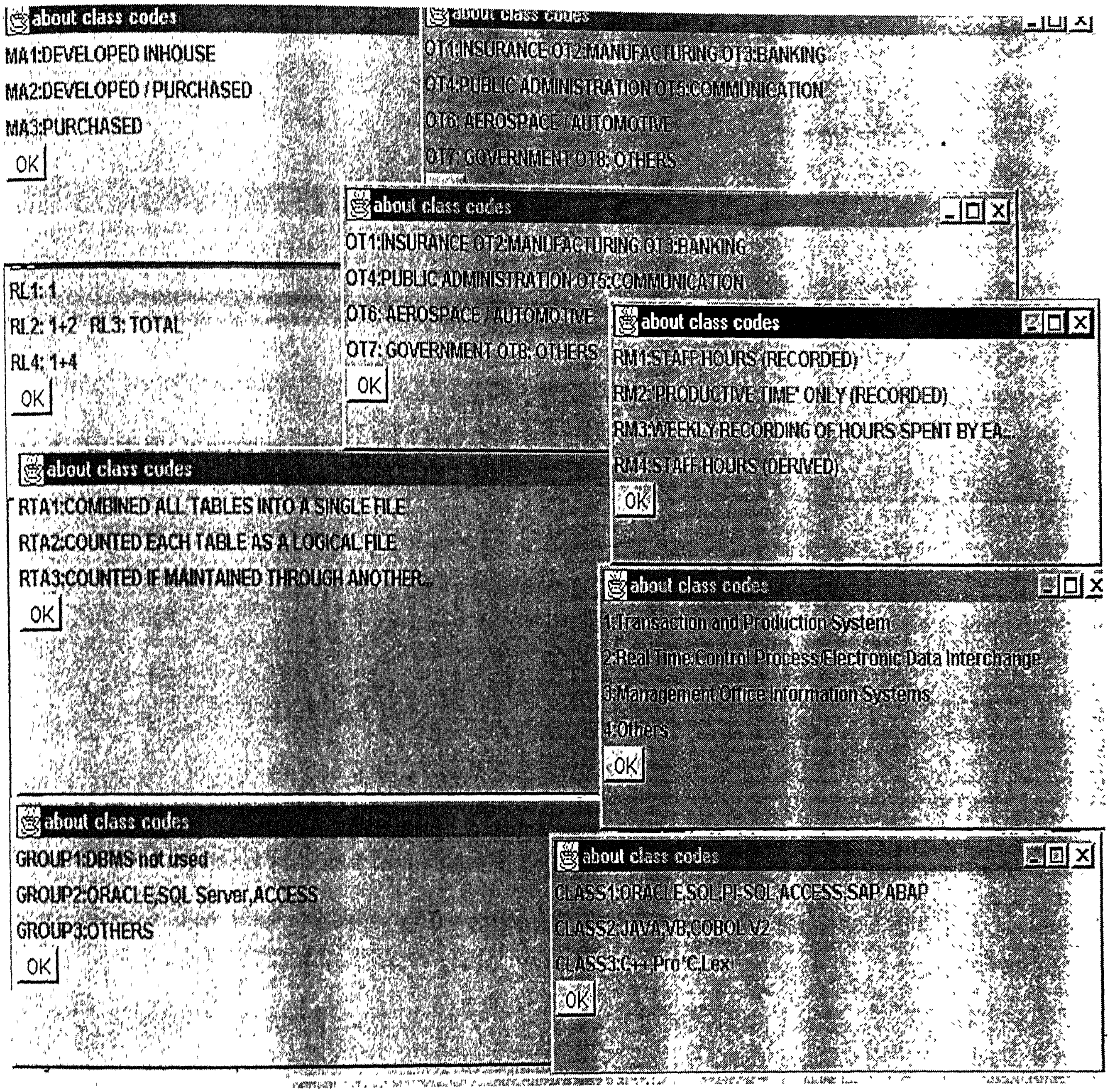
Virtual Machine Exp.

Language Exp.

Value Adj.Factor

Figure 4.4 Factor Input Screen-2

The various help menus regarding classification codes, to assist the user while he or she enters data are shown in Fig. 4.5.



MA1:DEVELOPED INHOUSE
MA2:DEVELOPED / PURCHASED
MA3:PURCHASED

OT1:INSURANCE OT2:MANUFACTURING OT3:BANKING
OT4:PUBLIC ADMINISTRATION OT5:COMMUNICATION
OT6: AEROSPACE / AUTOMOTIVE
OT7: GOVERNMENT OT8: OTHERS

OT1:INSURANCE OT2:MANUFACTURING OT3:BANKING
OT4:PUBLIC ADMINISTRATION OT5:COMMUNICATION
OT6: AEROSPACE / AUTOMOTIVE
OT7: GOVERNMENT OT8: OTHERS

RL1: 1
RL2: 1+2 RL3: TOTAL
RL4: 1+4

RM1:STAFF HOURS (RECORDED)
RM2:'PRODUCTIVE TIME' ONLY (RECORDED)
RM3:WEEKLY RECORDING OF HOURS SPENT BY EA...
RM4:STAFF HOURS (DERIVED)

RTA1:COMBINED ALL TABLES INTO A SINGLE FILE
RTA2:COUNTED EACH TABLE AS A LOGICAL FILE
RTA3:COUNTED IF MAINTAINED THROUGH ANOTHER...

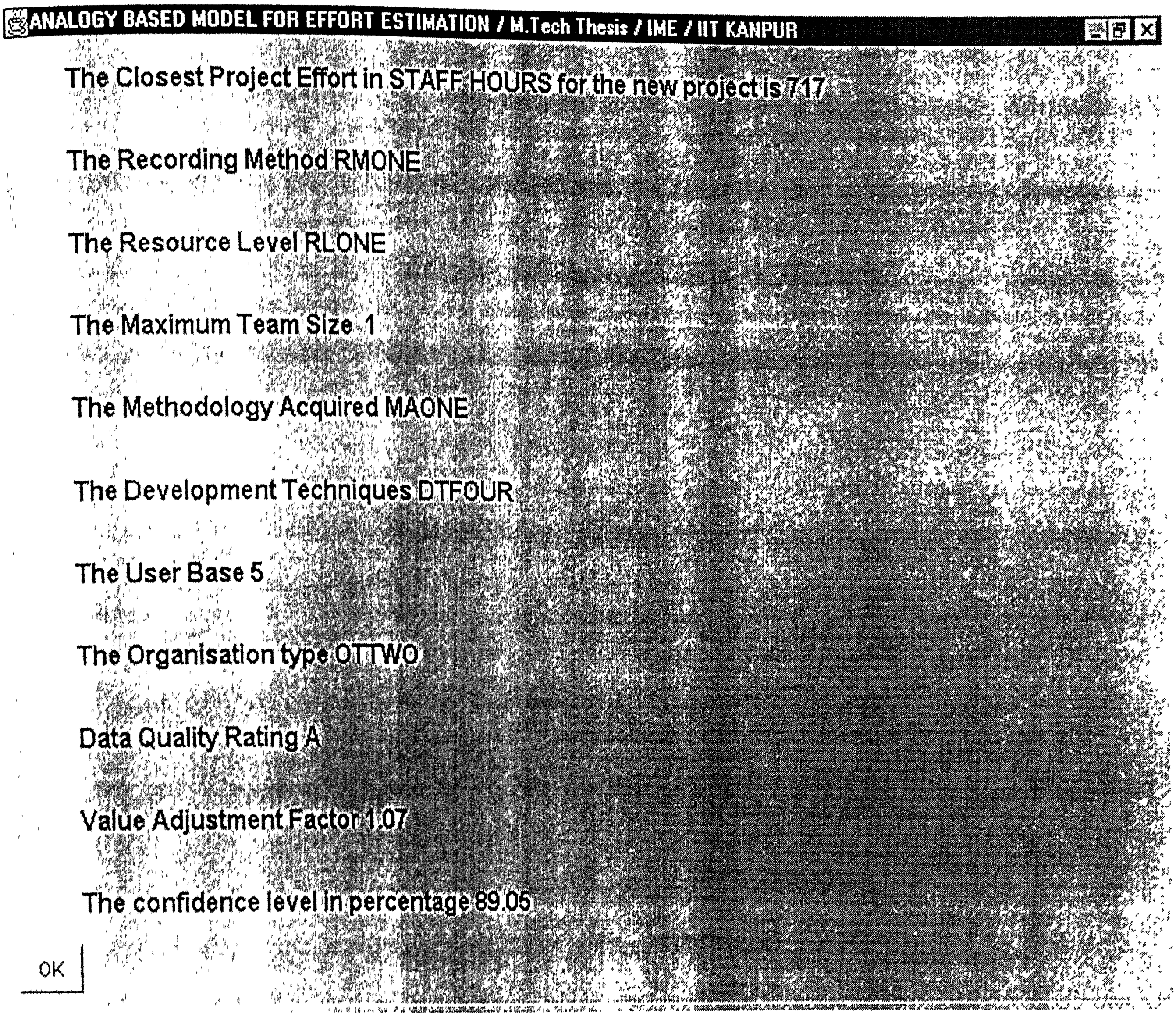
1:Transaction and Production System
2:Real Time/Control Process/Electronic Data Interchange
3:Management/Office Information Systems
4:Others

GROUP1:DBMS not used
GROUP2:ORACLE,SQL Server,ACCESS
GROUP3:OTHERS

CLASS1:ORACLE,SQL,PI-SQL,ACCESS,SAP ABAP
CLASS2:JAVA,VB,COBOL V2
CLASS3:C++,Pro*C,Lex

Figure 4.5 Help menus of class codes

The various features of the nearest and the next nearest analogous projects are provided to the user. These output screens are shown in Fig. 4.6 and 4.7.



ANALOGY BASED MODEL FOR EFFORT ESTIMATION / M.Tech Thesis / IME / IIT KANPUR

The Closest Project Effort in STAFF HOURS for the new project is 717

The Recording Method RMONE

The Resource Level RLONE

The Maximum Team Size 1

The Methodology Acquired MAONE

The Development Techniques DTFOUR

The User Base 5

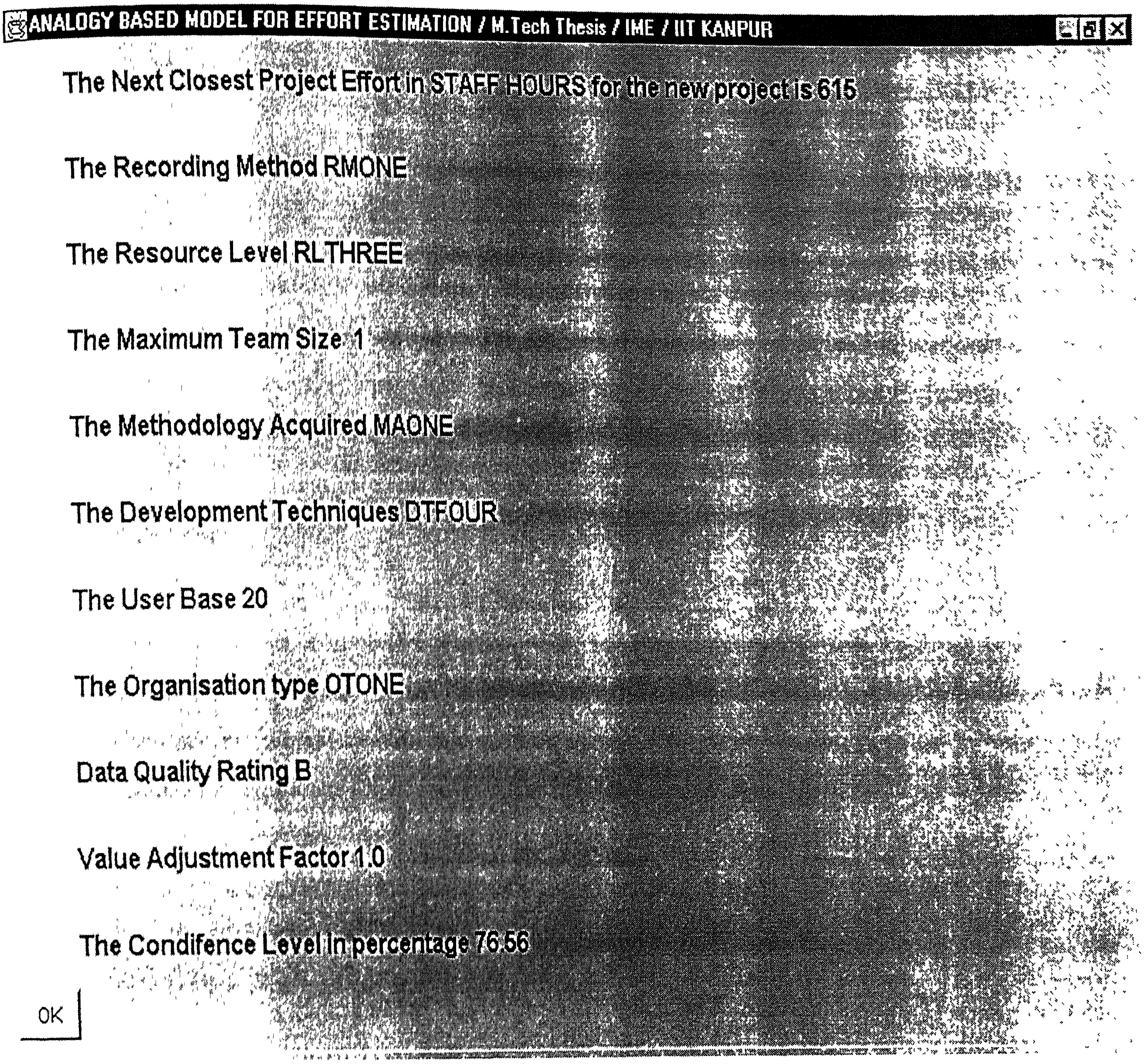
The Organisation type OTTWO

Data Quality Rating A

Value Adjustment Factor 1.07

The confidence level in percentage 89.05

Figure 4.6 Output Screen showing the nearest analogous project features



ANALOGY BASED MODEL FOR EFFORT ESTIMATION / M.Tech Thesis / IME / IIT KANPUR

The Next Closest Project Effort in STAFF HOURS for the new project is 615

The Recording Method RMONE

The Resource Level RLTHREE

The Maximum Team Size 1

The Methodology Acquired MAONE

The Development Techniques DTFOUR

The User Base 20

The Organisation type OTONE

Data Quality Rating B

Value Adjustment Factor 1.0

The Condifence Level in percentage 76.56

Figure 4.7 Output Screen showing the next nearest analogous project features

Chapter 5
Testing and Results

The model was tested for its validity. The effort estimates given by the following four versions of the model were compared for analysis. For the purpose of testing we took one by one 400 completed projects from the data set as the new project without including that project in the data set during search for the two nearest analogies.

5.1 Version 1

In this version, the completed projects with missing data fields were ignored while comparing with the new project. Analysis with this method makes use of only those observations that do not contain any missing values. With only 4% of the values missing at random in each of the factors, we lost 25% of the observations with list-wise deletion. Our view for this method is that this may result in many observations being deleted but may be desirable as a result of its simplicity.

The results obtained are summarized in Fig.5.1, Fig.5.2 and Fig.5.3.

Table 5.1 shows a sample of tested data points:

ACTUAL EFFORT	NEAREST ANALOGY	NEXT NEAREST ANALOGY	FUNCTION POINTS	MRE FOR THE NEAREST	MRE NEXT NEAREST
2150	2184	2867	342	1.58	33 35
1194	1124	1231	73	5 86	3 10
1835	1876	1231	236	2 23	32 92
4529	3934	2974	297	13 14	34.33
6381	7365	8716	193	15 42	36.59
7365	8716	6381	329	18.34	13 36
1876	2867	2184	174	52 83	16 42
1824	1876	1835	72	2 85	0.60
1231	1658	1194	95	34 69	3.01
*737	1824	1124	**71**	**147.49**	**52.51**
420	737	436	77	75 48	3 81

1922	1451	2184	143	24 51	13 63
***11514**	**18297**	**32760**	**1177**	**58.91**	**184.52**
1658	1835	737	282	10 68	55 55
2867	3360	1451	185	17 20	49 39
436	448	737	77	2 75	69 04
2040	1835	1876	156	10 05	8 04
***73501**	**32760**	**18297**	**1306**	**55.43**	**75 11**
3934	3360	4529	194	14 59	15 12
***18297**	**11514**	**32760**	**1290**	**37.07**	**79.05**
900	943	737	158	4.78	18 11
3360	2974	3934	326	11 49	17 08
1039	1139	1231	85	9.62	18 48
1723	1451	1876	248	15 79	8 88
32760	18297	11514	3460	44 15	64 85

Table 5.1: Version 1 A sample of tested data points

The " * " marked test projects are interesting to look at. The MRE for these projects is very high. The MMRE of the estimation on the test data was 41.95 for the nearest analogy and 48.26 for the next nearest analogy.

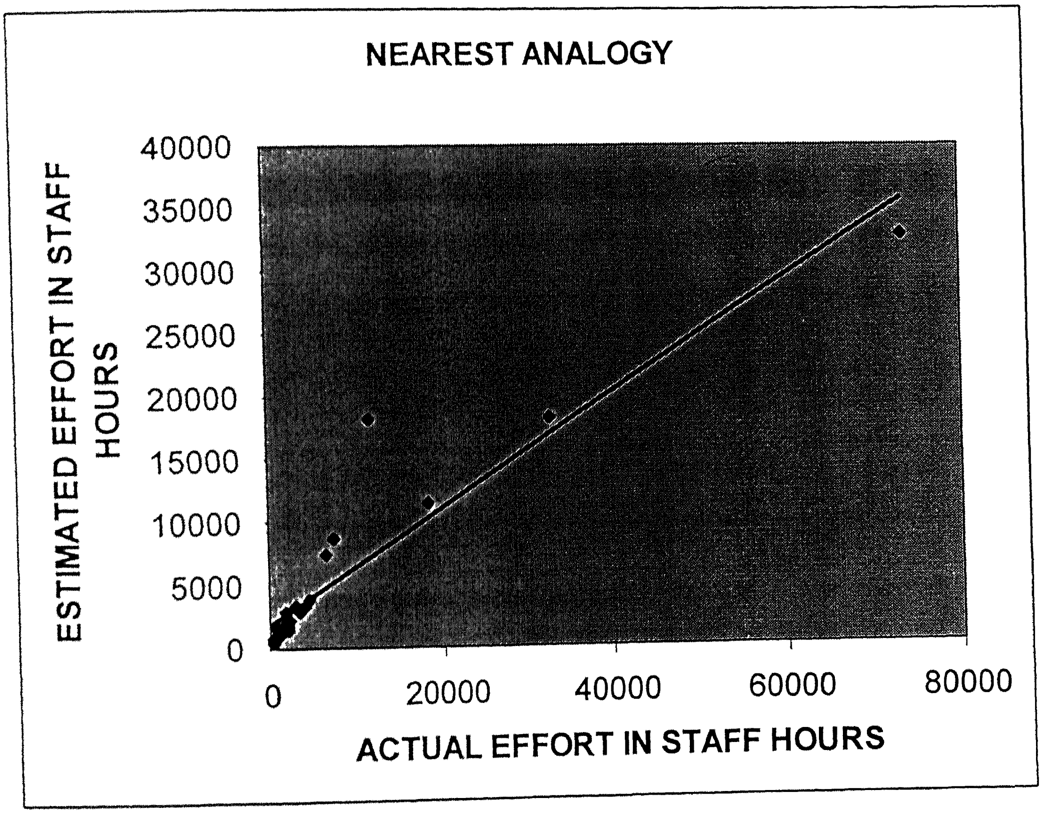


Fig. 5.1: Version 1 Actual Effort versus Estimated Effort for the nearest analogy

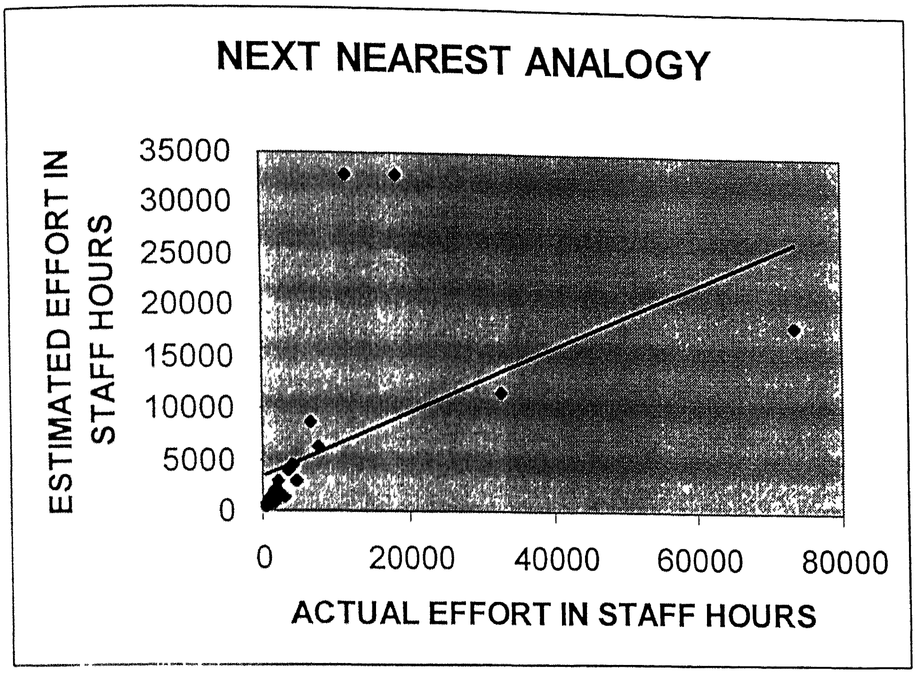


Fig. 5.2: Version 1 Actual Effort versus Estimated Effort for the next nearest analogy

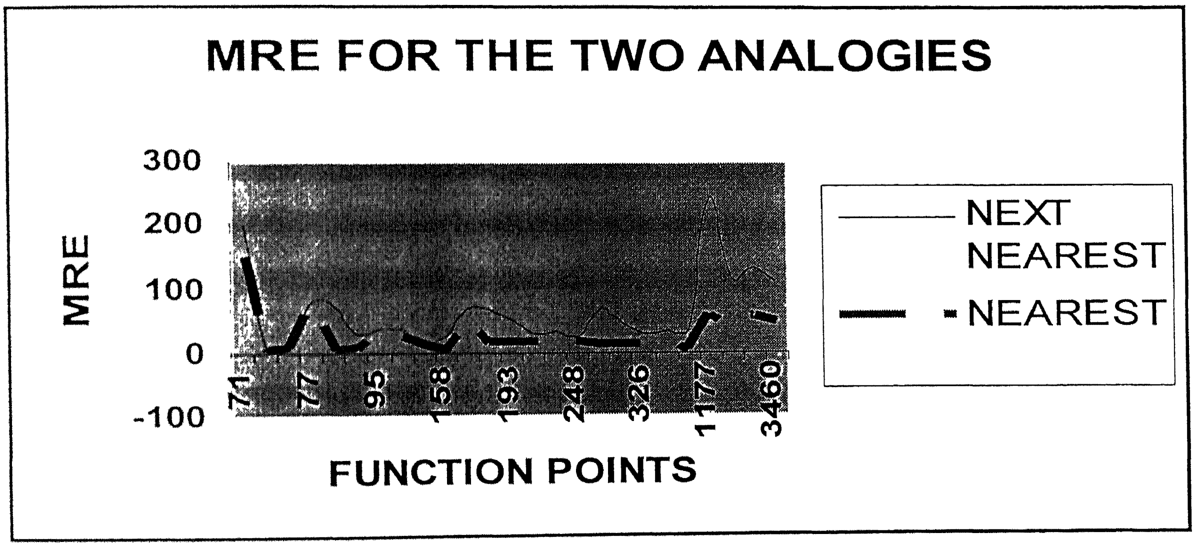


Fig. 5.3: Version 1 MRE for the two analogies

The analysis of Fig. 5.1 and Fig. 5.2 brought us to the conclusion that the list wise deletion method is appropriate only when there are small amounts of missing data and when the data is missing randomly. Due to loss of data of several completed projects, the estimates made by finding analogies can be quite misleading. They can give misleading information about any new project if its close completed projects have been deleted in list-wise deletion method. For example the "*" marked test projects in the above table lack the appropriate analogous projects in the data file dealt by the model. The abnormality in the Fig. 5.3 is found at the projects marked with asterisk. These are examples of misleading information due to insufficient data points. Also, it is interesting to note that the MRE for the two analogies do not show any specific trend with the increase in function points. The reason again is the basic characteristic of an analogy model, that is, the presence of analogous size projects in the data set is a must to provide better results. Fig.5.3 shows that the nearest analogy for list-wise deletion is more reliable than the next nearest analogy.

5.2 Version-2.1

The results of the first version were not satisfactory. Our next approach was to replace the missing field with the maximum value of that field. The results obtained are summarized in Fig. 5.4, Fig. 5.5 and Fig 5.6. The MMRE of the estimation on the test data was 31.78 for the nearest analogy and 32.29 for the next nearest analogy.

Table 5.2 shows a sample of tested data points:

ACTUAL EFFORT	NEAREST ANALOGY	NEXT NEAREST ANALOGY	FUNCTION POINTS	MRE FOR THE NEAREST	MRE NEXT NEAREST
2150	2184	2095	342	1 58	2 56
1194	1124	1231	73	5 86	3 10
1835	1876	1732	236	2 23	5 61
4529	3934	3935	297	13 14	13 12
6381	7365	8716	193	15 42	36 59
7365	8716	6381	329	18 34	13 36
1876	2867	2184	174	52 83	16 42

1824	1876	1835	72	2 85	0 60
1231	1658	1194	95	34 69	3 01
737	765	805	71	3 80	9 23
420	737	436	77	75 48	3 81
1922	1451	2184	143	24 51	13 63
11514	18297	17657	1177	58 91	53 35
1658	1835	737	282	10 68	55 55
2867	3360	1451	185	17 20	49 39
436	448	737	77	2 75	69 04
2040	1835	1876	156	10 05	8 04
73501	32760	18297	1306	55 43	75 11
3934	3360	4529	194	14 59	15 12
18297	22345	17657	1290	22 12	3 50
900	943	737	158	4 78	18 11
3360	2974	3934	326	11.49	17 08
1039	1139	1231	85	9 62	18 48
1723	1451	1876	248	15 79	8 88
32760	18297	22345	3460	44 15	31 79

Table 5.2: Version 2.1: A sample of tested data points

Its apparent from Table 5.1 and Table 5.2 that the estimation for the asterisk marked projects in the previous version has been considerably improved in this version of the model. The reason for this improvement is that the number of completed projects which are now dealt by the model in searching for the analogies is high.

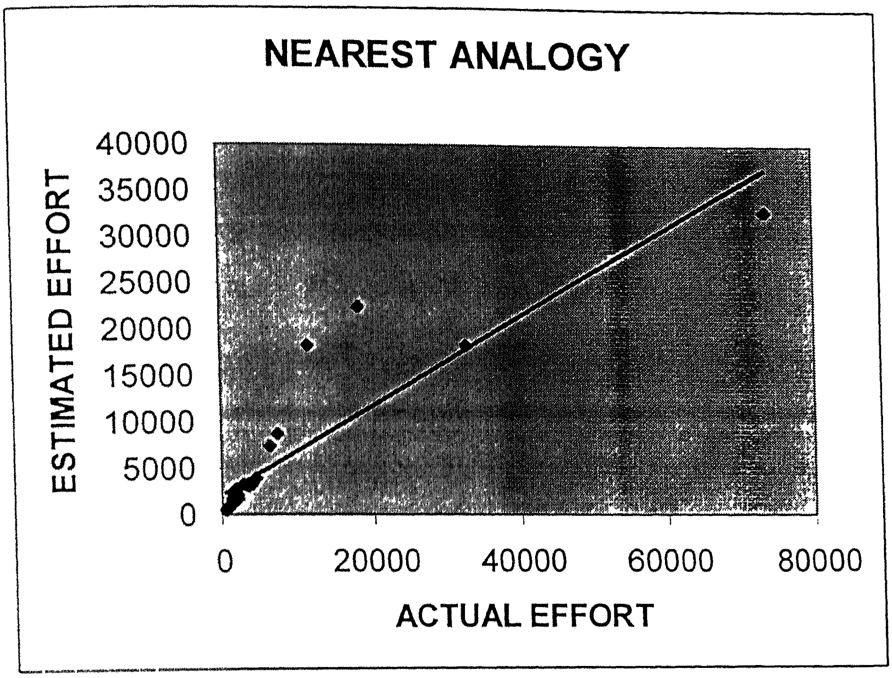


Fig. 5.4. Version 2.1 Actual Effort versus Estimated Effort for the nearest analogy

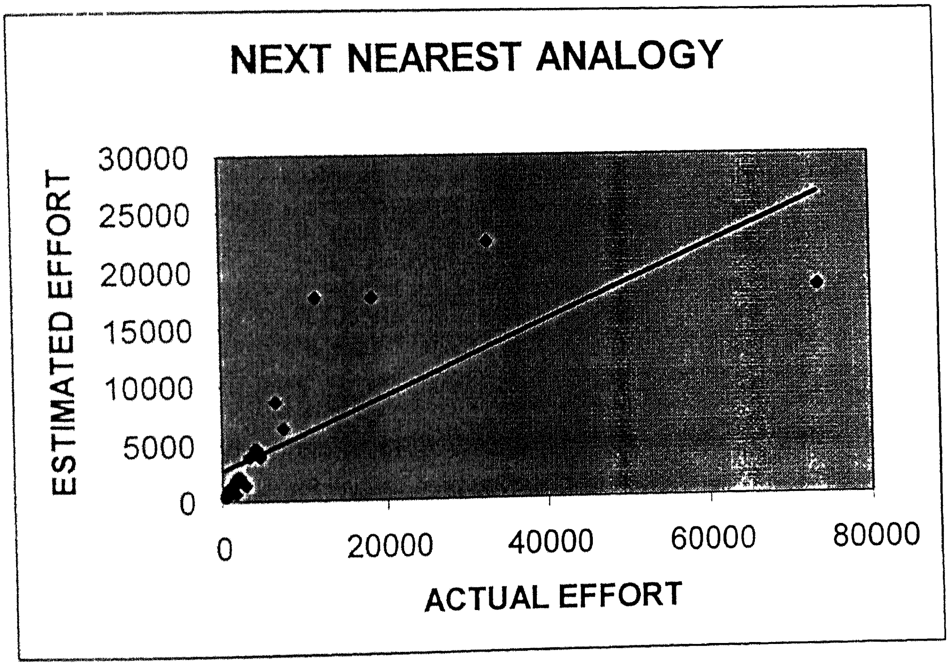


Fig. 5.5: Version 2.1 Actual Effort versus Estimated Effort for the next nearest analogy

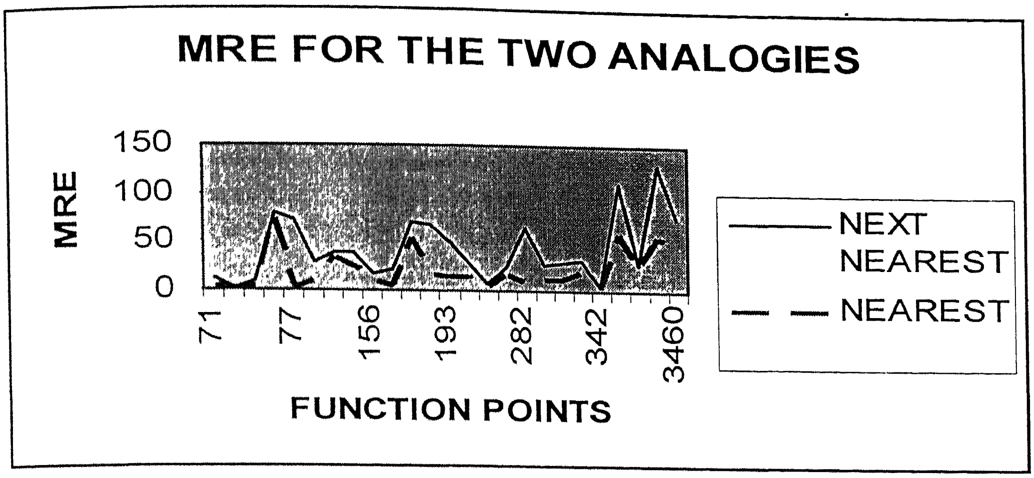


Fig 5.6: Version 2.1 MRE for the two analogies

Fig. 5.6 shows that the nearest analogy for this version is more reliable than the next nearest analogy.

.

5.3 Version 2.2

Our next approach was to replace the missing field with the minimum value of that field. The results obtained are summarized in Fig.5.7, 5.8 and 5.9. The MMRE of the estimation on the test data was 47.69 for the nearest analogy and 38.31 for the next nearest analogy.

Table 5.3 shows a sample of tested data points:

ACTUAL EFFORT	NEAREST ANALOGY	NEXT NEAREST ANALOGY	FUNCTION POINTS	MRE FOR THE NEAREST	MRE NEXT NEAREST
2150	2184	1876	342	1 58	12 74
1194	1124	737	73	5 86	38 27
1835	1876	1732	236	2 23	5 61
4529	3934	3935	297	13 14	13.12
6381	7365	8716	193	15 42	36 59
7365	8716	6381	329	18 34	13 36

1876	2867	2184	174	52 83	16 42
1824	1876	1194	72	2 85	34 54
1231	1876	1194	95	52 40	3 01
737	765	605	71	3 80	17 91
420	737	436	77	75 48	3 81
1922	1876	2184	143	2 39	13 63
11514	18297	17657	1177	58 91	53 35
1658	1835	605	282	10 68	63 51
2867	3360	1451	185	17 20	49 39
436	448	737	77	2 75	69 04
2040	1835	1876	156	10 05	8 04
73501	32760	22345	1306	55 43	69 60
3934	3360	4529	194	14 59	15 12
18297	22345	17657	1290	22 12	3 50
900	943	1194	158	4 78	32 67
3360	2974	3934	326	11 49	17 08
1039	1139	1231	85	9 62	18 48
1723	1451	1876	248	15 79	8 88
32760	18297	22345	3460	44 15	31 79

Table 5.3: Version 2.2: A sample of tested data points

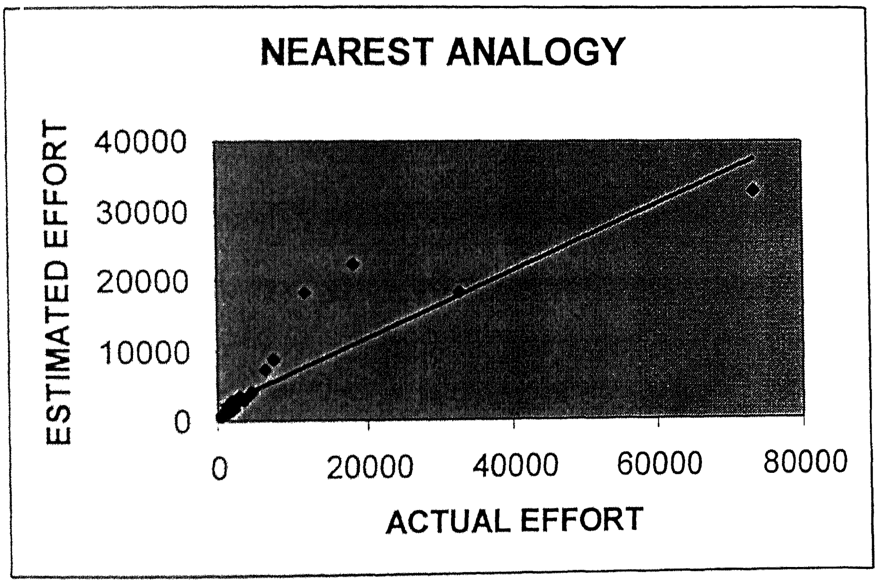
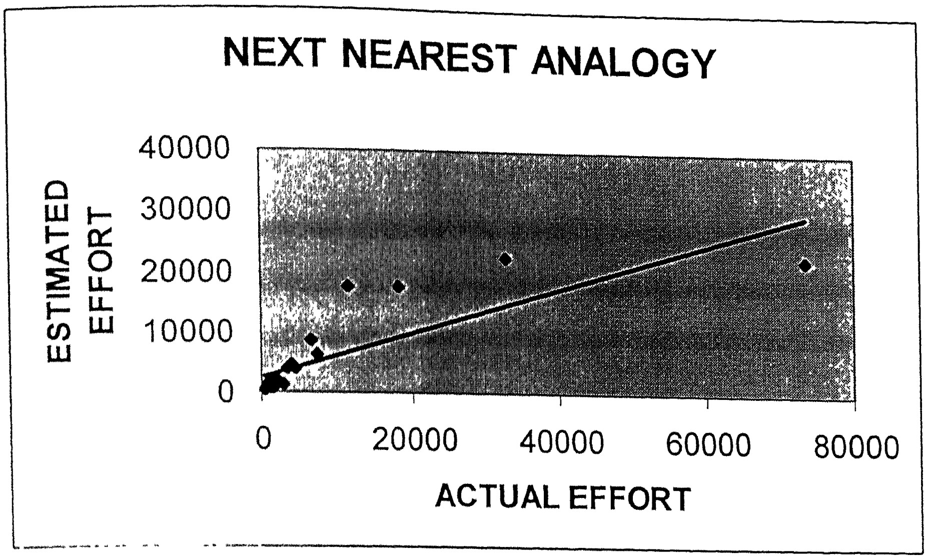


Fig. 5.7: Version 2.2 Actual Effort versus Estimated Effort for the nearest analogy



. Fig. 5.8: Version 2.2 Actual Effort versus Estimated Effort for the next nearest analogy

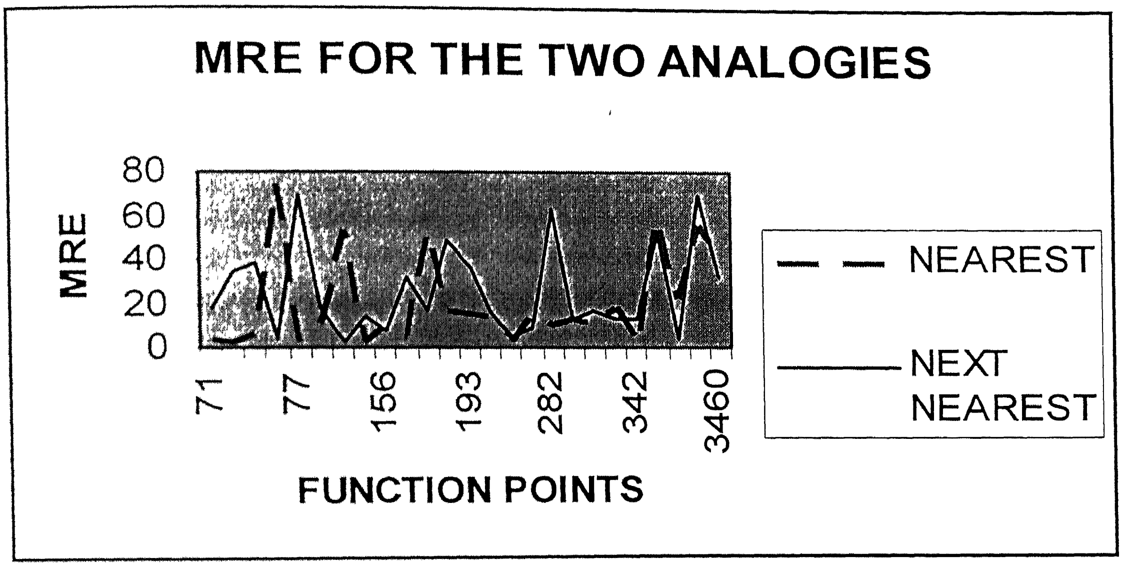


Fig. 5.9: Version 2.2 MRE for the two analogies

Again there is no considerable improvement as compared to version 2.1. The MMRE for both the analogies showed hike. The reason for such behavior is due to the characterization of the completed projects in the data file. Replacing a missing value with the minimum or maximum field value has random effect in the search for the analogies. We therefore do not recommend the replacement of the missing data as we did in the Version 2.1 and Version 2.2.

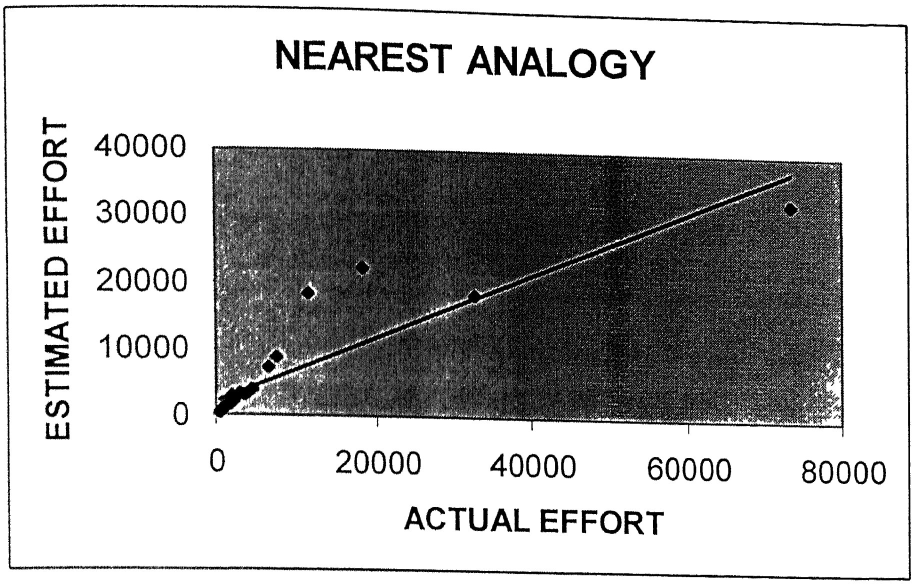
5.4 Version 3

Our next approach was to replace the missing data fields according to the Hot-Deck Euclidean Imputation method. The results obtained are summarized in the following graphs. The MMRE of the estimation on the test data was 21.57 for the nearest analogy and 24 98 for the next nearest analogy.

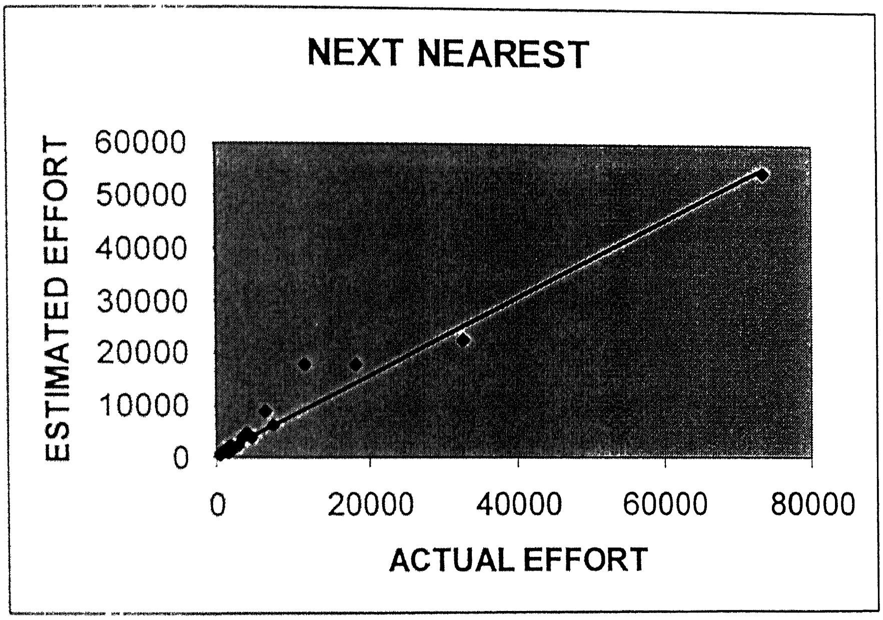
Table 5.4 shows a sample of tested data points:

ACTUAL EFFORT	NEAREST ANALOGY	NEXT NEAREST ANALOGY	FUNCTION POINTS	MRE FOR THE NEAREST	MRE NEXT NEAREST
2150	2184	1876	342	1 58	12 74
1194	1124	1231	73	5 86	3 10
1835	1876	1732	236	2 23	5 61
4529	3934	3935	297	13 14	13 12
6381	7365	8716	193	15 42	36 59
7365	8716	6381	329	18 34	13 36
1876	2867	2184	174	52 83	16 42
1824	1876	1194	72	2 85	34 54
1231	1124	1194	95	8 69	3 01
737	765	605	71	3.80	17 91
420	677	436	77	61 19	3 81
1922	1876	2184	143	2 39	13 63
11514	18297	17657	1177	58 91	53 35
1658	1835	605	282	10 68	63 51
2867	3360	2184	185	17 20	23 82
436	448	605	77	2 75	38 76
2040	1835	1876	156	10 05	8 04
73501	32760	54659	1306	55 43	25 64
3934	3360	4529	194	14 59	15 12
18297	22345	17657	1290	22 12	3 50
900	943	1194	158	4 78	32.67
3360	2974	3934	326	11 49	17 08
1039	1139	1231	85	9 62	18 48
1723	1451	1876	248	15 79	8 88
32760	18297	22345	3460	44 15	31 79

Table 5.4: Version 3 A sample of tested data



.Fig. 5.10: Version 3 Actual Effort versus Estimated Effort for the nearest analogy



. Fig.5.11: Version 3 Actual Effort versus Estimated Effort for the next nearest analogy

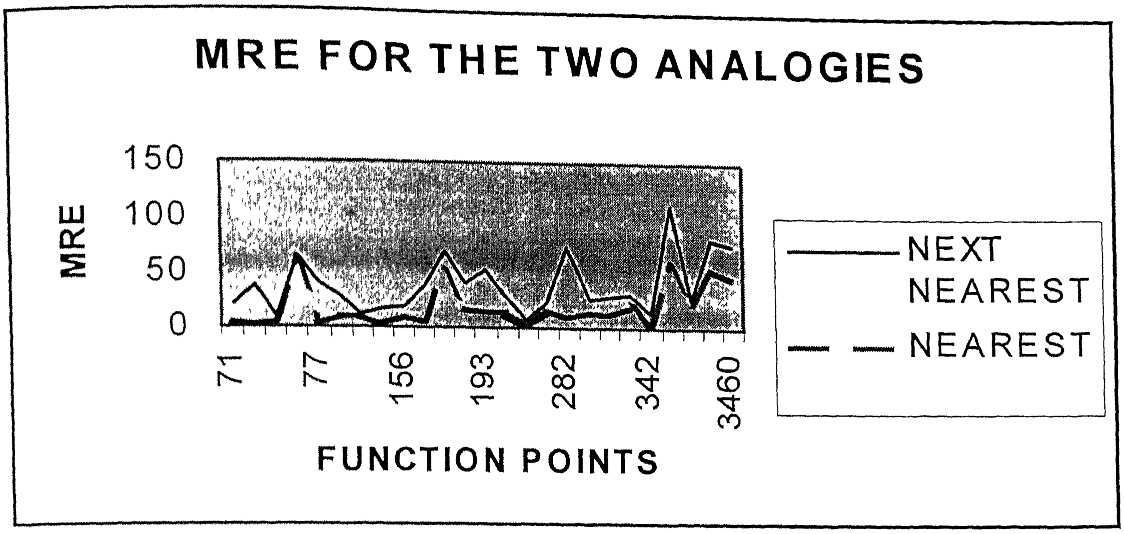


Fig. 5.12: Version 3 MRE for the two analogies

Fig.5.12 shows that the nearest analogy for Version 3 is more reliable than the next nearest analogy.

5.5 Inference

The comparison of the four versions of the model is summarized in the Table 5.5:

	Approach for Sparse Data	MMRE for the Nearest Analogy	MMRE for the Next Nearest Analogy	Recommendation
Version 1	List-Wise Deletion	41.95	48.26	Not Recommended
Version 2.1	Replacing with the Maximum value	31.78	32.29	Not Recommended
Version 2.2	Replacing with the Minimum Value	47.69	38 31	Not Recommended
Version 3	Hot-Deck Imputation	21.57	24.98	Recommended

Table 5.5: Comparison of four versions of the model

It is quite essential to note here that not one approach can ever give satisfaction on arriving at estimates. We worked on the four versions of the model and got the result to prefer Version 3. Though, we admit that it is quite possible that a good estimation for a particular new project can be got from any of the above methods. It all depends on the presence of the good analogies of the new project in the data set. It is therefore very important to strengthen the data file over time with all kinds of projects accomplished in the organization. Our recommendation for Version 3 is for the data set we have used. We leave grounds to the other researches to test the validity of this recommendation on other data sets.

Chapter 6

Conclusion and Future Scope

Accurate estimation of software project effort at an early stage in the development process is a significant challenge for the software engineering community. This work has described a technique based upon the use of analogy. We have compared the various versions of the model for dealing with the sparse data On the basis of the results obtained, we recommend that Hot-Deck imputation method (Version 3) gives appropriate analogies. We provide user with the information regarding two nearest projects. Our approach allows users to assess the reasoning process behind a prediction by identifying the nearest analogous projects thereby increasing, or reducing their confidence in the prediction

We encourage for the future research on the model. Its validity can be tested on other datasets as well. The effect of CMM level improvement of an organization on its historical data can also be studied. Our hypothesis is that the effort for a new project should decrease with the hike in the CMM level. It will be an interesting work to assess the factor by which effort decreases in such cases.

We finally conclude that not one effort estimating technique can ever produce satisfactory estimates. The hybrid of algorithmic and analogy based model should work with good precision.

References

1. Albrecht, A.J. and Gaffney, J. R., "Software function, source lines of code, and development effort prediction: a software science validation", IEEE Trans. on Software. Engg., 9(6), pp639-648, 1983.

2. Bate, Roger, Dorothy , Kuhn, Curt ,Wells, et al, "A System Engineering Capability Maturity Model, Version 1.1,", Software Engineering Institute, CMU Sei-95-MM-0003, www.sei.cmu.edu 1995.

3. Boehm, Barry ,W. ,"Software Engineering Economics", Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

4 Boehm, B., et. al., (1997),"COCOMO II Model Definition Manual", Version 1.4, University of Southern California, Los Angeles, CA

5. Chulani98 - "Calibrating Software Cost Models Using Bayesian Analysis", Chulani,Sunita, Boehm,B, and Steece, Technical Report, USC-CSE-98-508, Jun '98. To appear in IEEE Transactions on Software Engineering, Special Issue on Empirical Methods in Software Engineering.

6. Donald, C., McDermid, "Software Engineering for Information Systems", Great Britain, Blackwell Scientific Publications, 1990

7. Humphrey ,W.H., "A Discipline for Software Engineering", The Addition-Wesley object Technology New York, 1995.

8. Humphrey, W.H.,"Managing the Software Process", Addison-Wesley, 1989.

9. Jalote, P., "An Integrated Approach to Software Engineering", New York, Springer, 1997.

10. Jalote, P. , "CMM in practice: Process for executing Software Projects at Infosys", India, Addison Wesley Longman, 2000.

11 Kevin ,S, Khaled, El, and Madhavji, Nazim, "Software Cost Estimation with Incomplete Data", IEEE Trans.Software Engg. Vol. 27, No. 10,pp. 890,oct.2001.

12. Kitchenham, B, "The Problem with Function Points", IEEE Software, 14(2), pp. 29-31, 1997.

13 Matson, Jack E., Barrett ,Bruce E., and Mellichamp, Joseph M. (April 1993), "Software Development Cost estimation Using Function Points", IEEE Log No. 9216525.

14. Mukhopadhyay ,and Kekre, S, "Software Effort Models for the Early Estimation of Process Control Applications", IEEE Trans. Software Engg. Vol. 18,no. 10,pp. 915-924,oct.1992.

15. Park, R. (1992), "Software Size Measurement: A Framework for Counting Source Statements", CMU/SEI-92-TR-20, Software Engineering Institute, Pittsburgh.

7. Pressman, R.S. "Software Engineering: A Practitioner's Approach", New Delhi, McGraw-Hill International Editions, 1997

16. Putnam, Lawrence H., "A General Empirical Solution to the Macro Software Sizing And estimating Problem", IEEE Trans.Software Engg. Vol. Se-4, No. 4,pp. 345,july 1978.

17. Shepperd, Martin and Schofield, Chris (February 1997), "Estimating Software Project Effort Using Analogies", IEEECS Log No. 104091.

18 Shepperd, Martin and Cartwright, Michelle (January 2001), "Predicting with Sparse Data", IEEECS Log No. 114698.

19. Strike, Kelvin, Emam, Khaled and Madhavji Nazim (June 2000), "Software Cost estimation with Incomplete Data", Log No. 111188.

20 USC-CSE97 - "COCOMO II Model Definition Manual," Center for Software Engineering, Computer Science Department, University of Southern California, Los Angeles, CA. 90007,www.sunset.usc.edu.

21. Walker, Royce "Software Project Management: a Unified Framework", New Delhi, The Addition-Wesley object Technology Series, 1998..

22. www.ecfc.u-net.com

23. www.euclid.ii.metu.edu.tr

24. www.java.sun.com

25. www.qpmg.com

26. www.saspin.org

27. www.sei.cmu.edu

28. www.sunset.usc.edu

Appendix

(I) CMM

The Capability Maturity Model (CMM) for Software describes the principles and practices underlying software process maturity and is intended to help software organizations improve the maturity of their software processes in terms of an evolutionary path from ad hoc, chaotic processes to mature, disciplined software processes. The CMM is organized into five maturity levels:

1) **Initial.** The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.

2) **Repeatable.** Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

3) **Defined.** The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

4) **Managed.** Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

5) **Optimizing.** Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Predictability, effectiveness, and control of an organization's software processes are believed to improve as the organization moves up these five levels. While not rigorous, the empirical evidence to date supports this belief.

Except for Level 1, each maturity level is decomposed into several key process areas that indicate the areas an organization should focus on to improve its software process.
The key process areas at Level 2 focus on the software project's concerns related to establishing basic project management controls. They are Requirements Management, Software Project Planning, Software Project Tracking and Oversight, Software Subcontract Management, Software Quality Assurance, and Software Configuration Management.

The key process areas at Level 3 address both project and organizational issues, as the organization establishes an infrastructure that institutionalizes effective software engineering and management processes across all projects. They are Organization Process Focus, Organization Process Definition, Training Program, Integrated Software Management, Software Product Engineering, Inter-group Coordination, and Peer Reviews.

The key process areas at Level 4 focus on establishing a quantitative understanding of both the software process and the software work products being built. They are Quantitative Process Management and Software Quality Management.

The key process areas at Level 5 cover the issues that both the organization and the projects must address to implement continual, measurable software process improvement. They are Defect Prevention, Technology Change Management, and Process Change Management.

Each key process area is described in terms of the key practices that contribute to satisfying its goals. The key practices describe the infrastructure and activities that contribute most to the effective implementation and institutionalization of the key process area.

(II) Software Project Management Steps

The waterfall model, as stated in the introduction, is an engineering model designed to be applied to the development of software. The idea is the following: there are different stages to the development and the outputs of the first stage "flow" into the second stage and these outputs "flow" into the third stage and so on.

There are usually five stages in this model of software development:

1. ***Requirements analysis and definition.*** In this stage the requirements of the "to be developed software" are established. These are usually the services it will provide, its constraints and the goals of the software. Once these are established they have to be defined in such a way that they are usable in the next stage. This stage is often preceded by a feasibility study or a feasibility study is included in this stage. The feasibility study includes questions like: should we develop the software, what are the alternatives? It could be called the conception of a software product and might be seen as the very beginning of the life cycle.
2. ***System and software design***. In this stage the established requirements, flowing from the first stage, are identified as software or hardware requirements. The software requirements are then translated in such a way that they can be readily transformed into computer programs.
3. ***Implementation and unit testing.*** This is the stage where the computer programs are created. Each program is called a unit, and unit testing is the verification that every unit meets its specification.
4. ***System testing.*** All the units are combined and now the whole is tested. When the combined programs are successfully tested the software product is finished.
5. ***Operation and maintenance.*** Most software products include this stage of the development. It involves correcting errors that have gone undetected before, improvement and other forms of support. This stage is part of the life cycle of a software product, and not of the strict development, although improvements and fixes can still be considered as "development".

These steps are the main stages. There are also sub-stages, within each stage, but they differ from project to project. For example for management purposes the requirements stage is divided in a feasibility study, an outline requirements definition, a design study and a requirements specification stage.

It is also possible that certain software projects require the adding of an extra stage all together, or the splitting of one in two stages. However all the different waterfall models have the same underlying idea; the idea that one stage provides outputs which can be used as the input for the next stage. There thus is a linear flow amongst the stages. The progress of the software development, using the waterfall model, is thus easy to find out. A common way to look at the outputs of a certain stage and see whether or not they are finished in time, thus seeing how far the overall progress is.

There are also activities which are performed at every stage of the software development. These are documentation, verification and management. Documentation is intrinsic to the Waterfall model for it is document driven, as most of the outputs are documents. Verification, not only is a part of implementation & unit testing and system testing, but it is also part of all the other stages in the form of walk through, reviews and the like. Management involves the tailoring of the waterfall model to fit individual processes, managing the human resources (i.e. the people) and managing the rules and the protocol on how the output is formalized, who accesses what and other managing tasks.

Finally, it has to be noted that the software development process is not as linear as it seems. When errors in later stages are found, they are often fed back to a previous stage and the development is set back to that stage again. Since this is a managing nightmare, it often occurs that problems are ignored, left for later or programmed around. This feed back makes for a waterfall with information flowing both ways: down through the stages when something is made, and up through the stages when something goes wrong, or feedback is given. Also many processes are frozen when it is not yet the time to deal with them. This has led to the development of other, more flexible models.